Blockchain interoperability patterns

Guzmán Llambías[1,2], Laura González[1], and Raúl Ruggia[1]

[1]Facultad de Ingeniería, Universidad de la República
[2]Pyxis Research, Pyxis

March 18, 2024

Abstract

Design patterns are best practices for known problems in a specific context. Many patterns have been proposed in different domains, such as object-orientated programming, software architecture, and workflows, to name a few. However, blockchain interoperability is a recent area of work and, to our knowledge, no design patterns have been defined yet. The purpose of this work was to identify blockchain interoperability patterns that may exist in blockchain interoperability solutions. We identified six patterns through the observation of 35 interoperability solutions. A specification was built for each pattern using the Alexandrian template. The specification was evaluated with five semi-structured interviews with blockchain experts to collect data on the comprehension, completeness, and utility of the patterns. The results show that all interviewees identified the patterns. However, the pattern specification has different degrees of confidence in terms of clarity, completeness, and utility. Finally, all interviewees thought that the proposed patterns may be helpful to software architects in their first blockchain interoperability project.

Blockchain interoperability patterns

Guzmán Llambías*†, Laura González*, Raúl Ruggia*
* Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay
*{gllambi, lauragon, ruggia}@fing.edu.uy
†Pyxis Research, Pyxis, Montevideo, Uruguay
†guzman.llambias@pyxis.tech

***Abstract*—Design patterns are best practices for known problems in a specific context. Many patterns have been proposed in different domains, such as object-orientated programming, software architecture, and workflows, to name a few. However, blockchain interoperability is a recent area of work and, to our knowledge, no design patterns have been defined yet. The purpose of this work was to identify blockchain interoperability patterns that may exist in blockchain interoperability solutions. We identified six patterns through the observation of 35 interoperability solutions. A specification was built for each pattern using the Alexandrian template. The specification was evaluated with five semi-structured interviews with blockchain experts to collect data on the comprehension, completeness, and utility of the patterns. The results show that all interviewees identified the patterns. However, the pattern specification has different degrees of confidence in terms of clarity, completeness, and utility. Finally, all interviewees thought that the proposed patterns may be helpful to software architects in their first blockchain interoperability project.**

***Index Terms*—software architecture, blockchain, interoperability, design patterns**

I. Introduction

Design patterns are best practices for known problems in a specific context, and several design patterns have been specified on different domains (e.g. object oriented programming and software architecture). Blockchain domain is not the exception, and some patterns have been specified [1], [2].

Blockchain interoperability is a recent research area, as blockchains are silos of information that cannot interoperate with each other or other software systems [3], [4]. It is a challenge for a blockchain to communicate with another software system or accept data [5]. In recent years, the academic and industry community has proposed several solutions to deal with this challenge [4]. An observer can note that these solutions share some similarities with each other, and some of them may be identified as common practices to recurrent problems (i.e. patterns). However, to the best of our knowledge, no design patterns or best practices have been proposed to help software architects build these types of solution. Taking into account all this, this work poses the following research question: **What patterns can be identified from existing blockchain interoperability solutions?**

To answer this question, we analysed 35 blockchain interoperability solutions and identified six patterns. Each pattern was specified following the Alexandrian template [6] and qualitative methods were used to evaluate them. Semi-structured interviews were conducted to five blockchain experts to evaluate clarity, completeness and utility of the patterns.

The results show that there exist patterns in blockchain interoperability solutions. All interviewees identified two patterns or more and all patterns were identified by at least one interviewee. The results also show that the patterns had different levels of confidence related to clarity, completeness, and utility. For example, one pattern (i.e. Light Client) was not clear enough and not understood by most of the interviewees. All interviewees considered the patterns to be a good contribution, but could be improved and suggested some improvements to be more complete. However, all of them mentioned that all the problems they experienced were reflected in the patterns. However, they were not sure whether these patterns reflected all the interoperability problems that exist. All interviewees considered the patterns useful, and most of them believed that having these patterns should improve their decisions and "speed up" their development. Finally, all interviewees thought that the patterns may be helpful to software architects in their first blockchain interoperability project.

The main contributions of this paper are: 1) the identification of patterns in existing blockchain interoperability solutions by observing 35 blockchain interoperability solutions, 2) a specification of these patterns using the Alexandrian template, and 3) an evaluation using semi-structured interviews conducted to five blockchain experts to evaluate clarity, completeness, and utility of the patterns.

The remainder of the paper is organised as follows. Section II provides background concepts. Section III describes the research approach. Section IV describes the identified design patterns. Section V presents the results of the evaluation and discussion. Section VI analysed the related work. Finally, Section VII presents conclusions and future work.

II. Background

A. Blockchain

A blockchain is a distributed database over a network of nodes (or participants) structured as an ordered list of blocks [7]. Each block has a set of transactions and is linked to its predecessor blocks by a cryptographic hash. The hash of a block is built using the block data and the hash of the previous block. This protection enables nodes to notice if a block was modified. The modified blocks are discarded by these nodes. In practice, this behaviour provides data immutability to the blockchain. Once a transaction is committed, it cannot be modified or deleted.

Users of the blockchain can submit transactions, and each transaction must be signed with the user's private key. This ensures the integrity of the transaction and its ownership. Although transactions are signed with the user's private key, users use pseudonyms to submit transactions to the blockchain. This enables a certain level of anonymity for the users, although not full anonymity.

Blockchains use consensus protocol rules that nodes must follow to maintain the state of the database [7]. This behaviour enables the blockchain to operate without a centralised trusted third party to maintain the state of the database. These rules define how the nodes agree to consider a transaction valid and be committed to the blockchain. Two of the most popular consensus protocols are Proof-of-Work and Proof-of-Stake. Consensus protocols usually define a fee that users must pay to process submitted transactions. These fees motivate nodes to participate in the consensus protocol, as they are rewarded for validating the submitted transactions.

Some blockchains provide scripting capabilities to enable users to deploy autonomous software programs on the blockchain [7]. These programs are called Smart Contracts. Smart contracts can be triggered autonomously by any user by submitting a transaction to the blockchain. Smart contracts usually have local storage that enables them to hold data or digital assets such as cryptocurrencies. They can also invoke other Smart contracts or transfer digital assets from one user to another. Smart contracts can only use its local data as they follow the closed-world assumption of blockchain computation. In case they require data from external systems, they need to use Oracles to import these data. Oracles are a trusted third party that may have many forms. Oracles can be normal users that submit external data to the Smart Contract sor they can be specialised software that invoke the Smart Contracts to provide these data. Chainlink[1] is an example of an Oracle.

B. Blockchain interoperability

Wegner defines interoperability as "the ability of two or more systems to exchange information despite their differences in language, interfaces and execution platform" [8]. However, it is a challenge for blockchain systems to interoperate with each other or with other external software systems [5]. In general, blockchains are designed as information silos and do not provide interoperability capabilities. Furthermore, it is a challenge for a blockchain to accept data from another blockchain. In some scenarios, blockchains may trust external software systems and accept their data (e.g. Oracles). However, other scenarios are not straightforward, and blockchains must achieve consensus between each other on the truthfullness of the data exchanged [5].

Blockchain interoperability usually involves at least two blockchains, a source blockchain and a target blockchain. The source blockchain executes a local transaction that triggers a cross-chain transaction that spans the domain of the source blockchain into the domain of the target blockchain. As a result, a local transaction is submitted to the target blockchain. In some cases, the target blockchain trusts the source of the transaction and commits the transaction directly. However, in an untrusted scenario, the target blockchain must verify that the transaction was actually committed to the source blockchain and that it is reliable.

Blockchain interoperability solutions enable blockchain interoperability and the execution of cross-chain transactions. Several interoperability solutions have been proposed in recent years [3] [4]. Notary Scheme is a centralised solution in which a trusted third party monitors the source blockchain and submits transactions to the target blockchain. Sidechains defines a two-way peg mechanism to connect blockchains. The two-way peg mechanism enables bidirectional asset transfers between a mainchain (i.e. source blockchain) and a sidechain (i.e. target blockchain). Sidechains usually enhance the mainchain with new features, better performance, or lower costs. Sidechains may have their own consensus protocol and implementation and be totally different from the mainchain. As a result, the mainchain is isolated and protected from cryptographic breaks in the side chain. Relays are smart contracts deployed on each blockchain that verify the received transaction using Simple Payment Verification mechanisms. Atomic Swaps specify a protocol with steps that two users on two blockchains must follow for the atomic exchange of digital assets. Although Atomic Swaps are considered an interoperability solution by some literature, they do not follow Wegner's definition of interoperability, as they do not involve data exchange between blockchains. Enterprise relays are a trusted solution that involves two gateways, each gateway connected to a blockchain. They comprise a communication protocol that enables communication between gateways and cross-chain transactions. Gateways verify cross-chain transactions using verification proofs generated by other gateways. Finally, Blockchain of Blockchains is a sophisticated solution that involves a mainchain and several chains, which use the mainchain to execute cross-chain transactions.

Blockchain interoperability can be used to enable asset exchange, asset transfer, or data sharing between two or more blockchains. In an asset transfer scenario, an asset hosted on a source blockchain is moved to a target blockchain. The asset must be burnt or locked in the source blockchain, and a semantic equivalent asset must be created on the target blockchain. Burn/lock and creation of assets must be atomic tasks to maintain consistency between both blockchains. Otherwise, the asset may be duplicated. This problem is known as double spending the asset. The asset exchange scenario involves two users and two blockchains. User A hosts the asset A_1 in blockchain A, and user B hosts the asset B_1 in blockchain B. Users need to exchange assets, and as a result, user A hosts the asset B_1 on blockchain B and user B hosts the asset A_1 on blockchain A. This exchange must be performed atomically to avoid a user from keeping both assets. Finally,

[1]https://chain.link/

the data-sharing scenario involves a source blockchain that queries or sends data to a target blockchain.

III. Research approach

The research approach is depicted in Fig. 1 and started with a pattern mining phase. In this phase, observation and analysis of existing interoperability solutions were performed. The output was a set of candidate patterns that served as input to the second phase: pattern specification. In this second phase, patterns were specified following a pattern template and became the first draft of the patterns and output of the second phase. The process iterated through these two phases until a final version of the patterns was defined. The third phase performed the evaluation using qualitative methods, in particular, semi-structured interviews. This third phase returned to the pattern specification phase to improve the specification. The following subsections describe these phases in detail.

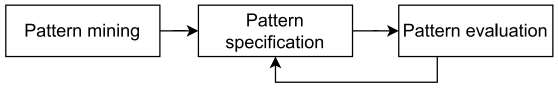

Fig. 1: Research approach

A. Pattern mining

This work considered the inductive approach for pattern mining, as it is considered a correct approach by the software pattern community [9]. In particular, following the Artifactual approach proposed by Kerth and Cunningham, that observes and analyses existing projects to mine patterns [10].

The literature review performed by Llambías et al. [4] served as an initial input, as it presented a feature-based classification of several blockchain interoperability solutions. During observation and analysis, the rule of three was applied to have an initial draft of the mined patterns. In the pattern community, the rule of three informally suggests that there should be at least three known uses of a solution to a problem, to consider that solution a pattern [9]. Later, this initial draft was refined by observation and analysis of other interoperability solutions reviewed by reference authors in the blockchain interoperability community [3]. Popular industry interoperability solutions were added in another iteration. Finally, through snowballing new interoperability solutions were analysed. Table I presents the trace of the interoperability solutions. Table II presents the recurrence of the patterns in the observed interoperability solutions.

Trace	Blockchain interoperability solution
Llambías et al. [4]	Polygon PoS Bridge, Cosmos, Weaver, RSK, Polkadot, Optimism, Hyperledger Cactus, BlockNET, ARK, YUI, Hermes, BTC Relay
Belchior et al. [3]	Quant, Peace Relay, Token Bridge
Popular	Solana Wormhole, BNB Chain Bridge, Arbitrum, StarkNET, Infura, Alchemy, Moralis
Snowball	Chain Bridge, ETH Near Relay, Block.io, GetBlock, BTC-Relay, zkBridge, Allbridge, zkSync, Celer Bridge, Kaleido, Chain API, Gravity Bridge, Quick Node

TABLE I: Interoperability solutions

B. Pattern specification

The patterns were specified following the Alexandrian template [6] and the guidelines proposed by Wellhausen and Fiesser [11]. Every specified pattern has a name that identifies it. They have a summary that resumes the pattern in a short description. It has a problem that describes a recurrent situation to be solved and helps to identify when to apply the pattern. It has a context that presents the preconditions so that the problem and solution are recurrent. Forces describe why the problem is difficult to solve. Patterns have a solution that explains how to solve the proposed problem in the specific context. They also have consequences that describe the benefits and liabilities to consider after the solution is applied. Patterns may be related to other patterns and they also have know usages that describe the occurrence of the pattern in the existing real world.

Pattern	Interoperability solution
Relayer	Polygon PoS Bridge, Cosmos, Weaver, RSK, Ark, YUI, Hermes, Token Bridge, Wormhole, BNB Chain Bridge, Arbitrum, StarkNet, Chain Bridge, ETH Near Relay, zkBridge, Celer cBridge, Gravity Bridge
API Gateway	Hyperledger Cactus, Quant, Infura, Alchemy, Moralis, Block.io, GetBlock, Kaleido, Chain API, Quick Node
Light Client	Cosmos, Weaver, RSK, Polkadot, BlockNet, YUI, BTC Relay, Peace Relay, BNB Chain Bridge, StarkNet, ETH Near Relay, BTC-Relay, zkBridge
Temporal transfer	Polygon PoS Bridge, RSK, Optimism Bridge, Wormhole, BNB Chain Bridge, Arbitrum, StarkNet, ETH Near Relay, Celer cBridge, Gravity Bridge
Permanent transfer	Polygon PoS Bridge, BNB Chain Bridge, Allbridge, Celer cBridge
Aggregator	Optimism Bridge, BNB Chain Bridge, StarkNet, zkBridge, zkSync, Gravity Bridge

TABLE II: Patterns invariance

C. Pattern evaluation method

To evaluate the patterns, this work used a qualitative evaluation method. In particular, semi-structured interviews were conducted to gather the opinion of blockchain experts about the proposed patterns. Semi-structured interviews are a suitable method for collecting qualitative data, as they provide an opportunity for discussion and exploration around a certain topic [12]. They were used to collect the opinion of blockchain experts to evaluate the understanding, completeness, and utility of the patterns.

The interview was designed based on the guidelines provided by Brinkmann and Kvale [12] and Turner [13]. The questions were grouped into four categories: identification, comprehension, utility, and completeness of the patterns. Table III presents the questions grouped by category.

A pilot interview was conducted to identify issues in its design. As a result, the questions were improved and refined. The timeframe of the interview was also adjusted.

Participants were selected using purpose and snowball sampling. Since blockchain interoperability is a recent topic, recruiting participants was not a straightforward task and required different recruitment strategies. First, a Linkedin post was published that described the work and called interested candidates to participate in the interview. This allowed us

Category	Questions
Identification	Did you identify any of the proposed patterns in your previous experiences or were all new to you?
Comprehension	What do you think of the clarity of the patterns? Would you change something to improve its comprehension, or do you think they are clear enough?
	What do you think was the most difficult part to understand or maybe not difficult?
Utility	How useful or not are these patterns for a blockchain developer starting his first interoperability project?
	What would have changed or not in your previous blockchain interoperability experiences if you have had these patterns?
Completeness	What do you think about the description of the patterns? Do they provide all the information you think is necessary to use them or should they provide more information around certain topics?
	Do you think there is an interoperability issue not reflected in the patterns that is necessary to include, or do you think all the aspects you know are covered?

TABLE III: Questions grouped by category

to recruit two interviewees. Second, an email was sent to a colleague referral that was accepted. Third, academic recruitment was performed at a computer science conference[2]. Snowballing was performed asking the recruited candidates for a referral interested in the topic. Snowballing allowed the recruitment of two interviewees. All candidates must have between two and five years of experience in blockchain-based development, seven years in the information technology area, and at least participation in one project in which they were required to interoperate two or more blockchains. Table IV presents the demographic information of the interviewees[3]. The recruitment time frame limited the number of total participants.

id	Role	IT	BCE	BCIP	EL	Nat.
Pilot	senior blockchain developer	10	8	1	Master Degree	Kosovo
1	blockchain company co-founder	7	3	1	PhD Student	Brasil
2	blockchain developer	11	3	1	System Analyst	Uruguay
3	Lead blockchain developer	18	5	+3	Under-graduate	Uruguay
4	Associate Professor	16	5	1	PhD	Italy
5	Technical lead	20	5	1	Computer Systems Degree	Uruguay

TABLE IV: Participants information.

The interviews were conducted in November 2023. Each participant received an email with the patterns[4], main topics to be covered in the interview, the expected duration, and the measures taken related to privacy and data confidentiality. All participants signed an informed consent to use their opinions and thoughts in this research. The interviews were carried out using Zoom or Microsoft Teams, depending on the language of the interview. Zoom was used for English speakers and Microsoft Teams for Spanish speakers. This decision was made primarily due to the automatic transcript capabilities of each tool. All interviews were recorded in video and audio. On average, the interviews had a duration of 73 minutes, in which the shortest took 50 minutes and the longest took 82 minutes. The interviewer took notes during each interview.

The raw data collected from the interviews were processed and then analysed. Data analysis was performed in parallel with data collection (i.e. interview execution) and followed an inductive approach. Codes and categories were progressively defined after each interview analysis [5]. Microsoft Word comments were used to code the transcripts.

After data analysis, the main conclusions of each interview were sent to the interviewee to confirm their opinions. All interviewees returned positive feedback about this resume.

The design, execution and data analysis of the interviews were performed by the main author of this article. The interview questions were designed by all the authors.

IV. Blockchain interoperability Design Patterns

This section describes a resume of the proposed patterns following the Alexandrian template. Table V presents a general overview and Fig. 2 the pattern relationships.

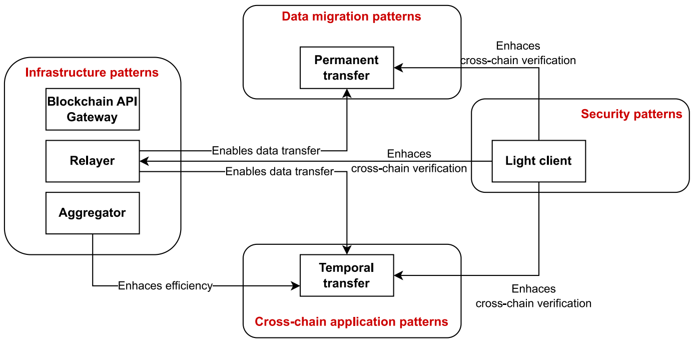

Fig. 2: Pattern relation overview

A. *Relayer pattern*

Summary: A third-party middleware (centralised or decentralised) called Relayer enables to send the state of an executed transaction from a source blockchain to a target blockchain. Blockchains may be homogeneous or heterogeneous. Fig. 3 presents a graphical representation of the pattern.

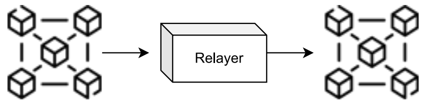

Fig. 3: Relayer pattern

Context: A target blockchain needs the state of an executed transaction on a source blockchain to perform a task (e.g. execute a smart contract).

Problem: How to send the state of an executed transaction from a source blockchain to a target blockchain.

[2]Enterprise Design, Operations and Computing

[3]IT = years of experience in information technologies, BCE = years of experience in blockchain based development, BCIP = number of blockchain interoperability projects, EL = Education level, Nat = Nationality.

[4]The original patterns sent to the interviewees can be found in (url to be published after reviewers-comments)

[5]The code book employed in this interview can be found at (url to be published after reviews to keep a double blind review)

Category	Pattern	Summary
Infrastructure	Relayer	A Relayer enables to send the state of an executed transaction from a source blockchain to a target blockchain.
	API Gateway	The API Gateway simplify the connectivity between traditional business applications and one or more blockchains through an API.
	Aggregator	The Aggregator listens to the execution of transactions on a source blockchain and packages a large set of these transactions into a compressed dataset. Then submits this dataset to a target blockchain as a unique transaction.
Security	Light Client	A smart contract on the target blockchain verifies the received data of a source blockchain and makes the data available to other smart contracts.
Data migration	Permanent transfer	An asset hosted on a blockchain is permanently transferred to another blockchain.
Cross-chain application design	Temporal transfer	An asset hosted on a source blockchain is temporary transferred to a target blockchain. That asset can be temporary used in the target blockchain until it is returned back to the source blockchain.

TABLE V: Interoperability patterns overview

Forces:

- **Communication:** Usually, blockchain technology does not provide a native mechanism to communicate directly with another blockchain. It is a hard task for a blockchain to interoperate with another blockchain.
- **Data format:** The source blockchain may use a different data format than the target blockchain.
- **Decentralisation:** Blockchain-based applications try to avoid using centralised components.
- **Performance:** Data must be available in a timely manner on the target blockchain to complete the task.
- **Costs:** the execution and storage of blockchain transactions usually imply costs for the user that submits the transaction. Computational processing in a blockchain also implies costs for the user.
- **Transaction signing:** Users must sign all transactions submitted to the blockchain.

Solution: Use a middleware component called Relayer that listens to the transactions committed on the source blockchain and sends them to the target blockchain. The Relayer enables point-to-point communication between the two blockchains. The Relayer can transform the transaction data format from the source blockchain data format to the target blockchain data format, so that the latter can process these data. The Relayer signs the transaction submitted to the target blockchain on behalf of the source blockchain.

Relayers may be developed for a specific scenario or may be general-purpose Relayers that can be reused in several scenarios. The first always create the same blockchain transactions or invoke the same smart contract on the target blockchain, and is built for a specific purpose of a business process. The second are smart Relayers that can create different blockchain transactions or invoke different smart contracts depending on the transaction content. Finally, the Relayer can be a native node of the blockchain tightly coupled with its behaviour or it can be provided by an external third party.

Consequences:

Benefits:

- **Communication:** The source blockchain can send the state of an executed transaction to a target blockchain.
- **Data format:** The target blockchain can understand the syntax of the received transaction.

Liabilities:

- **Communication:** The Relayer provides a point-to-point communication. In case the source blockchain needs to communicate with another blockchain, a new Relayer must be deployed to enable point-to-point communication between them.
- **Performance:** The Relayer may be a bottleneck and may not send the transaction state in a timely manner to the target blockchain.
- **Decentralisation:** In case the Relayer is centralised, it becomes a point of failure that needs to be monitored. Furthermore, a centralised component adds trust to the solution that may contradict the use of blockchain technology. A decentralised Relayer reduces this risk but adds more complexity, as there must be consensus among all the Relayer instances.
- **Costs:** The computational processing and storage of the received transaction on the target blockchain incur costs that must be paid by the Relayer.
- **Transaction signing:** The Relayer signs the transactions submitted to the target blockchain and acts on behalf of the source blockchain.

Related patterns: The Light Client pattern provides cross-chain transaction verification to a trusted Relayer.

Known uses: Wormhole, Weaver, Cosmos, Token Bridge.

B. API Gateway pattern

Summary: A third-party middleware called API Gateway simplifies the connectivity of one or more traditional business applications with one or more blockchains through an API. Fig. 4 presents a graphical representation of the pattern.

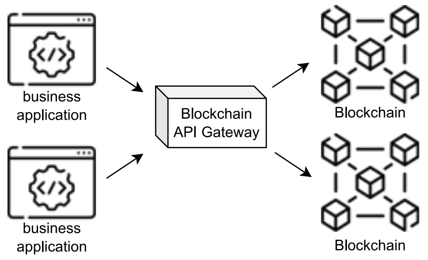

Fig. 4: API Gateway pattern

Context: A business application needs to communicate with one or more blockchains to complete its business process. For example, to submit transactions or query the blockchain.

Problem: The business application have to understand the low-level connectivity details to communicate with one or more blockchains.

Forces:

- **Blockchain heterogeneity:** Every blockchain has its own technical requirements to establish a connection to it.
- **Security:** Every blockchain requires a private key to sign the transactions submitted to the blockchain.
- **Infrastructure:** Business applications needs a blockchain node to communicate with the blockchain. To connect to N blockchains, a business application needs to host N blockchain nodes. One node for each blockchain.
- **Availability:** If the blockchain node is down, the business application cannot communicate with the blockchain.
- **Costs:** The blockchain node requires hardware and software costs that must be considered.
- **Decentralisation:** The blockchain operation and availability do not depend on individual decisions of its participants (i.e. decentralised). The business application only needs a blockchain node to communicate with it and does not require any authorisation or permissions.
- **Performance:** The blockchain node must handle the load generated by the business application.

Solution: Use a third-party middleware called API Gateway that exposes a high-level interface and enables a business application to communicate with one or more blockchains without requiring the application to understand the low-level technical details of the communication.

Consequences:

Benefits:

- **Blockchain heterogeneity:** business applications do not have to understand low-level details to connect with each blockchain. They use a high-level interface (e.g. http endpoints) to connect to the blockchain.
- **Blockchain node:** No blockchain node is required to connect with the blockchain.
- **Costs:** The owner of the business application must not afford hardware and software for each blockchain node and costs related to monitoring it.

Liabilities:

- **Security:** The business application may not own the private key to connect to each blockchain. The API Gateway may act on behalf of the business application.
- **Availability:** The API Gateway becomes a new point of failure in the solution. This component is not controlled by the business application and unpredictable down-times may affect its normal operation.
- **Costs:** API Gateways usually charge for the services provided and this cost must be considered.
- **Decentralisation:** The API Gateway is a centralised component provided by a third party that reduces the level of decentralisation. If the owner of the API Gateway unilaterally defines any change (e.g., cancelling the service), the business application might be affected.
- **Performance:** An additional intermediary adds an overhead in the communication. The API Gateway can be a communication bottleneck.

Related patterns: No related patterns were identified.

Known uses: Infura, Alchemy, Quant, GetBlock, Kaleido.

C. Light client pattern

Summary: A smart contract on the target blockchain stores a trusted consensus state of a source blockchain and provides operations to verify data against this state. Fig. 5 presents a graphical representation of the pattern.

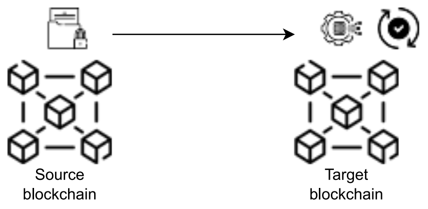

Fig. 5: Light client pattern

Context: A source blockchain sends data to a target blockchain using an interoperability mechanism (e.g. applying the Relayer pattern). Blockchains may use different consensus protocols, but support the same technical capabilities (e.g. same hash functions and signature algorithms).

Problem: How can smart contracts hosted on the target blockchain can use the received data reliably without a trusted third party.

Forces:

- **Data acceptance:** It is a challenge for a blockchain to accept external data from another blockchain. The target blockchain must verify that the received data are valid and have been committed to the source blockchain.
- **Data query:** In general, smart contracts cannot directly query the blockchain ledger. Smart contracts can only query data from its local storage or another smart contract.
- **Transaction storage:** Transactions committed to the blockchain require storage to store them.
- **Costs:** Usually, data storage and computational execution on the blockchain implies costs for the user that submitted the transaction. Smart contract invocation to another smart contract may also imply costs to the user.
- **Performance:** Transactions compete to be grouped into a block and recorded on the blockchain. The order in which transactions are sent to the blockchain may not be the same in which they are processed and recorded.
- **Transaction execution failure:** Validators (e.g. miners in Bitcoin) reject transactions with errors or failures. A block with rejected transactions is discarded and is not registered on the blockchain.

Solution: Use a smart contract on the target blockchain called Light Client, that receives data from the source blockchain and validates it using the attached proof. Some common data proofs include block headers and zero knowledge proofs. The Light Client provides functions to query the stored data. Light clients are usually implemented using smart contracts and use its local storage to store the received data.

Consequences:

Benefits

- **Data acceptance:** The target blockchain has verified data from the source blockchain that is available to be used by business process implemented on the target blockchain.
- **Data query:** Smart contracts hosted on the target blockchain can query the Light Client smart contract to use the received data.

Liabilities:

- **Transaction storage:** New data requires storage to be recorded on the target blockchain and may impact the requirements to host a node of the target blockchain.
- **Costs:** Submitting data and store it on the Light Client implies costs to the submitter. Querying the Light Client may also imply costs.
- **Performance:** Data verification on the Ligth Client is usually considered as a transaction on the target blockchain. This transaction must compete with other transactions to be included on a block. The processing and verification of this new transaction may not be immediately performed as it is received.
- **Transaction execution failure:** A failure processing a transaction caused by a bug in the Light Client, may cause to miss data from the source blockchain or affect the target Light Client reliability (e.g. false positives).

Related patterns: Relayer, Temporal and Permanent transfer patterns may use the Light Client to enable cross-chain transaction validation.

Known uses: BTC Relay, Cosmos, RSK, Weaver, YUI.

D. Temporal asset transfer pattern

Summary: An asset hosted on a source blockchain is temporary transferred to a target blockchain. That asset can be temporary used in the target blockchain until it is returned back to the source blockchain. Fig. 6 presents a graphical representation of the pattern.

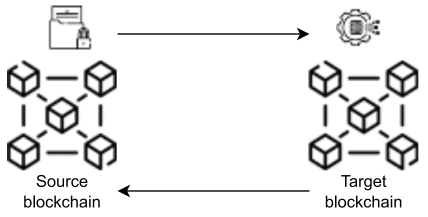

Fig. 6: Temporal transfer pattern

Context: Blockchains execute transactions with different costs and performance. In addition, blockchains have different characteristics and not every blockchain supports the same features. Blockchains have technical compatibility (e.g. same hash functions and signature algorithms) that enables them to semantically understand shared data. A user may want to temporary move an asset from a source blockchain to a target blockchain to have some benefit of these blockchain heterogeneity (e.g. execute transactions at less costs).

Problem: How an asset from a source blockchain can be temporary transferred to a target blockchain and returned. How can be returned back with all its state changes performed by the execution of transactions in the target blockchain.

Forces:

- **Data acceptance:** It is a challenge for a blockchain to accept external data from another blockchain. The target blockchain must verify that the received data are valid and have been committed to the source blockchain.
- **Costs:** The execution and storage of blockchain transactions usually imply costs for the user that submits the transaction.
- **Transaction execution failure:** A failure verifying a transaction by a blockchain validator (e.g. miners in Bitcoin) causes the transaction to be discarded.
- **Asset availability:** In a given moment, an asset can only be available for use in a single blockchain. In other case, there is the risk of double spending the asset.

Solution: The asset must be blocked on the source blockchain and create a semantically equivalent asset on the target blockchain. The locked asset cannot be used or changed. After the asset has been used on the target blockchain, it must be destroyed, and if and only if the asset was correctly destroyed, then the asset on the source blockchain must be unlocked.

Consequences:

Benefits:

- **Asset availability:** The asset is available to use in the target blockchain.
- **Costs:** Transactions may be executed at a lower cost or with higher performance in case the target blockchain enables it.

Liabilities

- **Costs:** Locking and unlocking the asset on the source blockchain and minting and destroying the asset on the target blockchain are operations that have costs to be considered.
- **Transaction execution failure:** A failure in the process of destroying the asset in the target blockchain may block the asset permanently on the source blockchain.
- **Data acceptance:** The asset must not be minted on the target blockchain until the target blockchain verifies the asset has been locked on the source blockchain. The asset must not be unlocked on the source blockchain until the source blockchain verifies it has been destroyed on the target blockchain.

Related patterns: The Tempral asset transfer uses the Relayer to send data from the source blockchain to the target blockchain. It can also use the Light Client pattern to enable cross-chain transaction validation when an untrusted or centralised Relayer is used.

Known uses: RSK, Polygon PoS Bridge, Wormhole.

E. Permanent asset transfer pattern

Summary: An asset hosted on a blockchain is permanently transferred to another blockchain. Fig 7 presents a graphical representation of the pattern.

Context: A user has an asset on a blockchain and does not want to use that blockchain anymore. This may happen in

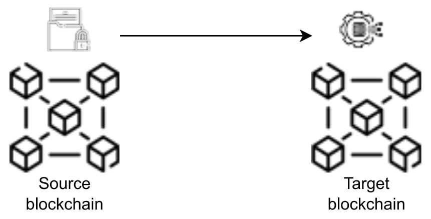

Fig. 7: Permanent transfer pattern

case another blockchain provides better features, less costs or higher performance, among other reasons. Blockchains have technical compatibility (e.g. same hash functions and signature algorithms) that enables them to semantically understand shared data.

Problem: How to permanently move an asset from a source blockchain to a target blockchain.

Forces:

- **Data acceptance:** It is a challenge for a blockchain to accept external data from another blockchain. The target blockchain must verify that the received data are valid and have been committed to the source blockchain.
- **Costs:** the execution and storage of blockchain transactions usually imply costs for the user who submits the transaction.
- **Transaction execution failure:** A failure verifying a transaction by a blockchain validator (e.g. miners in Bitcoin) causes the transaction to be discarded.
- **Asset availability:** In a given moment, an asset can only be available for use in a single blockchain. In other case, there is the risk of double spending the asset.

Solution: The asset in the source blockchain is destroyed and a new asset is created on the target blockchain that is semantically equivalent.

Consequences:

Benefits

- **Asset availability:** The asset is available to use at the target blockchain and is semantically equivalent to the destroyed asset. The latter is destroyed in the source blockchain and cannot be recovered.

Liabilities:

- **Data acceptance:** The asset must be minted on the target blockchain only when the target blockchain verifies the asset has been destroyed on the source blockchain.
- **Costs:** The destruction and minting of the asset have a cost for the user. In case the Relayer pattern is used, the Relayer may incur in additional costs for the user for the data transfer.
- **Transaction execution failure:** A failure in the minting process may imply a complete loss of the asset if the asset was destroyed on the source blockchain.

Related patterns: Uses the Relayer pattern for data transfer and the Light Client for cross-chain transaction validation.

Known uses: Polygon PoS Bridge, Allbridge, Celer cBridge.

F. Transaction aggregator pattern

Summary: A third party middleware called Aggregator listens to the execution of transactions on a source blockchain and packages a large set of these transactions into a compressed dataset. This compressed dataset is submitted as a unique transaction to a target blockchain. Fig 7 presents a graphical representation of the pattern.

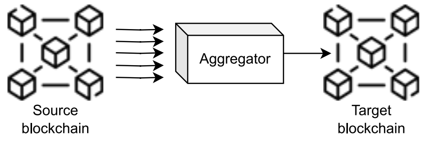

Fig. 8: Transaction aggregator pattern

Context: The source blockchain needs to send the state of a large set of executed transactions to a target blockchain.

Problem: How to efficiently send the state of a large set of transactions executed on a source blockchain to a target blockchain.

Forces:

- **Communication:** In general, blockchain technology does not provide a native mechanism for communicating directly with another blockchain. It is a hard task for a blockchain to interoperate with another blockchain.
- **Costs:** Each time a user submits a transaction to the blockchain, the user must pay a fee to process the transaction.
- **Performance:** Only a finite number of transactions can be included in a block. Processing a large set of transactions may be time consuming, as they may need to be processed in more than one block.
- **Transaction selection:** Transaction validators define which transactions must be included in a block based on the incentive they have to process the transaction. Some transactions of the set may not be processed until the validators defines to do so.
- **Data acceptance:** It is a challenge for a blockchain to accept external data from another blockchain. The target blockchain must verify that the received data are valid and have been committed to the source blockchain.
- **Decentralisation:** Blockchain technology is decentralised by nature. This design enables its correct operation besides individual decisions made by its participants.
- **Transaction storage:** Storage is required on the target blockchain to store the details of transactions sent by the source blockchain.

Solution: Use a component called Aggregator that listens to the recorded transactions on the source blockchain and groups them into a batch. The Aggregator digests this batch of transactions, creating a smaller piece of data called Rollup, with the resulting state of each transaction. This Rollup is a smaller piece of data and does not contain all the details of each transaction, just the resulting state after its execution on the source blockchain. Afterwards, the Aggregator sends the Rollup to the target blockchain as a unique transaction.

Consequences:

Pattern \Interviewee	1	2	3	4	5	**Total**
Relayer	x			x	x	**3**
API Gateway	x	x	x	x	x	**5**
Light Client				x		**1**
Temporal transfer	x	x		x	x	**4**
Permanent transfer				x	x	**2**
Aggregator	x	x	x			**3**

TABLE VI: Pattern identification by interviewee

Benefits:

- **Communication:** A source blockchain can notify the execution state of a large set of transactions to a target blockchain. This communication is enabled by the Aggregator.
- **Costs:** It is possible to process a large set of transactions in the target blockchain with less costs than processing them individually.
- **Performance:** It is possible to efficiently process a large set of transactions on the target blockchain in less time, compared to process them individually.
- **Transaction storage:** The record of the execution state at the target blockchain only requires the storage of a unique transaction with the execution state of the set of transactions executed on the source blockchain.

Liabilities:

- **Transaction selection:** The validators at the target blockchain may not have the right incentives to process the Rollup and the transaction may not be processed.
- **Decentralisation:** The Aggregator decides which transactions may be grouped together in a batch. If it is a centralised third party, it may be biased in the transaction selection and leave some transactions unattended.
- **Data acceptance:** A transaction verification mechanism must be defined to validate the Rollup on the target blockchain. Two examples are optimistic verification or zero-knowledge proofs verification.
- **Costs:** The Aggregator may charge additional costs for generating the batch. There are costs related to the execution of the Rollup on the target blockchain that need to be considered.

Related patterns: The Aggregtor uses the Light Client pattern for cross-chain verification. The Aggregator uses the Relayer to send data from the source to the target blockchain.

Known uses: Arbitrum, zkBridge, StarkNet and zkSync.

V. Evaluation

The patterns were evaluated through semi-structured interviews to assess clarity, completeness, and utility.

A. Results

Table VI presents the patterns identified by each interviewee. All interviewees identified the usage of two or more patterns in their previous experiences. All patterns were identified by at least one interviewee. All interviewees considered the patterns to be a good contribution to blockchain interoperability.

1) Clarity: The clarity of a pattern refers to the level of understanding of a pattern for the interviewee. All interviewees stated that they got a general understanding of the patterns, although four out of six interviewees (included pilot) did not understand the Light Client pattern or thought it was an Oracle pattern. After an explanation from the interviewer, two of the interviewees recognised the pattern and two did not know about the pattern before. An interviewee worked with the RSK bridge and understood the pattern because she recognised it in the known examples. All interviewees stated that they would get a global understanding of the patterns if the relationship between them were explained. Two interviewees mentioned that a pattern categorisation would also have helped. Two interviewees mentioned that the patterns were easy to understand. All interviewees suggested specific adjustments to improve the clarity of the patterns. Some of these suggestions were included in this work, while others are planned to be included in future work.

2) Completeness: The completeness of a pattern implies how complete the patterns are considering the information provided and the known-to-date interoperability problems of the interviewees. Use case examples showing how patterns may be used was the most mentioned improvement for completeness. Furthermore, two interviewees mentioned that the patterns required more technical details to implement them (e.g. a reference implementation). All interviewees mentioned that all the problems they had experienced in blockchain interoperability projects were reflected in the examples. However, some interviewees were not sure if these patterns reflected all existing interoperability problems. In particular, an interviewee suggested analysing the Blockchain of blockchains solution and analyse if a pattern could be mined. In addition to the previous comments, all interviewees suggested interesting improvements to enrich the patterns[6].

3) Utility: The utility of a pattern refers to the usefulness of a pattern for blockchain interoperability projects. All interviewees found the proposed patterns useful, although an interviewee mentioned that the patterns are useful for a designer audience and not much for a developer that needs to implement them. Three interviewees mentioned that these patterns may allow them to "make more conscious design decisions" and "speed them up", as patterns describe known problems and solutions. Two interviewees stated that these patterns would serve to learn about blockchain interoperability. Finally, all interviewees mentioned that these patterns would have helped them in some way during their first interoperability experience.

B. Discussion

One of the main key findings of this study was that all interviewees identified two or more patterns, and all patterns were identified by at least one interviewee. The study strongly shows that the API Gateway and the Temporal transfer patterns are widely applied patterns in the sample population, as all

[6]A list of all the suggested improvements can be found at (url to be published after reviews to keep a double blind review)

interviewees identified the former, and most of them identified the latter pattern. A similar trend was found for two patterns (i.e. Relayer and Aggregator), although it was less strong. This last trend led to the consideration that these two patterns are likely to be used in the software industry. Most of the interviewees did not understand correctly one of the patterns (i.e. Light Client), and some of them were misguided to think about Oracles. This shows that the Light Client pattern was not clear enough. However, we consider that this pattern needs to be rewritten rather than discarded. Table II shows that there exists a common behaviour around cross-chain transaction verification. We consider that this common behaviour is likely to be specified as a pattern. Finally, the results show a trend that these patterns may help software architects and designers to build blockchain interoperability solutions, as all interviewees mentioned in the interview.

Overall, the results of this study partially answered the proposed research question **What patterns can be identified from existing blockchain interoperability solutions?**. This study can claim that there are blockchain interoperability patterns and that some of them were discovered with different levels of confidence. However, we do not pretend with this study to cover all blockchain interoperability patterns. The discovery of patterns is far from complete. Second, this study shows that the proposed patterns may help blockchain developers during their first experience with blockchain interoperability. Third, remarks on the clarity of some patterns showed that the evaluation method is robust and allowed us to detect patterns that do not meet the expected clarity to be understood by the sample population.

This study opens many future research directions. First, interoperability solutions can be modelled using these patterns and start working around specifications instead of code. Second, new guidelines can be proposed for software architects to design their interoperability solutions based on these patterns. Finally, model-driven development tools can be built following these patterns to ease the development of blockchain interoperability solutions.

C. Threats to validity

External validity Although most of the interviewees identified the proposed patterns, this cannot be generalised to the global blockchain community. Clearly, a low number of interviewees is not enough to draw a general conclusion. Furthermore, since 50% of the interviewees were from Uruguay, this could have biased the results to only a geographical region.

Internal validity The communication skills of the interviewer has a direct effect on the gathered data through semi-structured interviews [12]. This may limit the amount and quality of the collected data. To mitigate this threat, a pilot was conducted to rehearse the interview, and each interviewee confirmed a resume of the main conclusions of the interview. This allowed us to confirm the results of each interview. Furthermore, data were analysed as they were collected by confirming the results with the interviewee in the interview itself, as recommended by Brinkmann and Kvale [12].

Coding and data analysis were performed by the main author of this work, which could have biased the final results.

Patterns were specified by the main author and supervised by the other authors, and no other validation was performed. To mitigate pattern writing accuracy, the guidelines provided by Wellhausen and Fiesser [11] were followed.

The patterns were extracted through the observation of blockchain interoperability solutions documentation. It is possible that the misinterpretation and biases of the researcher could have influenced the results.

VI. Related work

Montgomery et al. [14] presented four blockchain interoperability patterns: Ledger transfer, Atomic Swap, Ledger interaction, and Ledger entry point coordination. We consider the ledger transfer pattern to be a combination of the Temporal and Permanent transfer patterns. Although both patterns share some similarities, we consider it better to treat them differently, as they have different consequences and behaviour. Montgomery et al. considered the Atomic Swap, however, we decided not to consider it, as it does not satisfy the definition of interoperability. They involve two independent transactions that are coordinated by two users in each blockchain, and the data do not span from a source blockchain to a target blockchain. The patterns described by Montgomery et al. provide a brief description of each pattern. Our approach followed the Alexandrian template, providing more information and an evaluation using semi-structured interviews.

The Weaver team presented three patterns to integrate distributed ledgers [15]. This is an interesting approach, as they cover distributed ledgers and not only blockchain ledger structures. Some similarities may be found with our patterns, but little information is provided to perform an accurate comparisson. The Consensus-based integration between ledgers pattern refers to the fact that a source ledger communicates its consensus view to a target ledger. A consensus view is an agreed representation of the state of the ledger by all members of the ledger. This pattern seems to be a combination of our Relayer and Light client patterns. The Standard API integration between application pattern refers to build an API on top of the distributed ledger to enable the exchange of its state and share similarities with the Relayer pattern. Finally, the Single enterprise participating in multiple networks pattern refers to a third-party that coordinates the exchange of state between the ledgers. It shares similarities with the Notary Scheme (see Section II and none of our patterns is similar to this proposal. It would be interesting to analyse it in the future. Unfortunately, this work does not provide any evaluation to compare with our results.

Xu et al. [1] presented a collection of design patterns for blockchain-based applications with four categories: Interaction with External World, Data Management, Security and Structural Patterns of Contract. Lieu et al. [16] proposed twelve design patterns for blockchain-based self-sovereign identity systems to help understand and apply self-sovereign identity to software design. They grouped these patterns into three groups: key management, decentralised identifier management, and credential design. Mühlberger et al. [2] proposed four Oracle patterns: Pull-based inbound oracle, Push-based inbound oracle, Pull-based outbound oracle and Push-based outbound oracle. These works used a similar template pattern to our work, but no evaluation was performed that allowed us to compare them. Mühlberger et al. used qualitative evaluation methods to evaluate the patterns.

Finally, Wöhrer and Zdun [17] proposed a set of patterns for smart contract development on Ethereum. Through a Multivocal Literature Review, they mined eleven design patterns grouped in five categories: Action and Control, Authorisation, Lifecycle, Maintenance and Security. For each pattern, they provided a problem, a solution, and illustrative code examples. The pattern mining process performed by this work took a different approach from ours. They analysed the code of Ethereum smart contracts hosted on open source repositories, while our analysis was performed on the documentation of existing interoperability solutions. A code analysis of existing interoperability solutions can complement our work to provide more details about the implementation of our patterns.

Overall, several authors proposed patterns around blockchain and blockchain interoperability with different levels of specification and evaluation. On the one hand, academic studies around blockchain patterns provide rich specifications using well-known pattern templates, but lack of evaluation methods to confirm their usage. On the other hand, blockchain interoperability patterns are proposed by the industry using a brief introduction to the patterns, but provide little information to serve as best practice guidelines. There is some overlap between existing work and our paper, which helps to confirm the correct direction in the proposed patterns.

VII. Conclusions and future work

This study identified six blockchain interoperability patterns by observing 35 blockchain interoperability solutions. These patterns were specified using the Alexandrian template and evaluated using qualitative methods. Semi-structured interviews were conducted with five blockchain experts to evaluate their comprehension, completeness, and utility. As a result, it was possible to partially answer the proposed research question **What patterns can be identified from existing blockchain interoperability solutions?** All interviewees identified two or more patterns, and all patterns were identified by at least one interviewee. This result led us to confirm that a small sample of the software industry identified the patterns. Furthermore, the results showed different levels of confidence. Some patterns (i.e. API Gateway and Temporal transfer patterns) were clearly identified by the sample population, while other patterns (i.e. Relayer and Aggregator patterns) were likely identified. A pattern (i.e. Light Client) was not clear enough and could not be understood by most of the sample population. This remark highlights the robustness of the evaluation method in detecting patterns that do not meet the expected clarity to be understood by the sample population. Finally, the sample considered that the proposed patterns may help blockchain developers speed up their development in their first blockchain interoperability experience.

Future work includes improving the patterns through suggestions from the interviewees. In addition, use quantitative methods to evaluate quantitative characteristics of the patterns. Finally, the development of guidelines to help software architects design interoperability solutions based on these patterns.

Acknowledgment

Guzmán Llambías was supported by Pyxis. The research that gives rise to the results presented in this publication received funding from the Agencia Nacional de Investigación e Innovación under the code POS_NAC_2022_4_174476. I am very grateful to all the interviewees who participated and to Diego Vallespir for his support with the interviews. Icons made by Freepik from www.flaticon.com

References

[1] X. Xu, C. Pautasso, L. Zhu, Q. Lu, and I. Weber, "A pattern collection for blockchain-based applications," in *Proceedings of the 23rd European Conference on Pattern Languages of Programs*, ser. EuroPLoP '18, New York, NY, USA, 2018. [Online]. Available: https://doi.org/10.1145/3282308.3282312

[2] R. Mühlberger, S. Bachhofner, E. Castelló Ferrer, C. Di Ciccio, I. Weber, M. Wöhrer, and U. Zdun, "Foundational oracle patterns: Connecting blockchain to the off-chain world," in *Business Process Management: Blockchain and Robotic Process Automation Forum*, Seville, Spain, Sep. 2020, pp. 35–51.

[3] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *ACM Comput. Surv.*, vol. 54, no. 8, pp. 1–41, Oct. 2021. [Online]. Available: https://doi.org/10.1145/3471140

[4] G. Llambías, L. González, and R. Ruggia, "Blockchain interoperability: a feature-based classification framework and challenges ahead."

[5] B. Pillai *et al.*, "Cross-blockchain technology: Integration framework and security assumptions," *IEEE Access*, vol. 10, pp. 41 239–41 259, Apr. 2022. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3167172

[6] C. Alexander, *The timeless way of building*. Oxford, United Kingdom: New york: Oxford university press, 1979, vol. 1.

[7] X. Xu, I. Weber, and M. Staples, *Architecture for blockchain applications*. New York, NY, USA: Springer Cham., 2019.

[8] P. Wegner, "Interoperability," *ACM Comput. Surv.*, vol. 28, no. 1, p. 285–287, mar 1996. [Online]. Available: https://doi.org/10.1145/234313.234424

[9] C. Kohls and S. Panke, "Is that true...? thoughts on the epistemology of patterns," in *Proceedings of the 16th Conference on Pattern Languages of Programs*. New York, NY, USA: Association for Computing Machinery, 2009. [Online]. Available: https://doi.org/10.1145/1943226.1943237

[10] N. Kerth and W. Cunningham, "Using patterns to improve our architectural vision," *IEEE Software*, vol. 14, no. 1, pp. 53–59, Feb. 1997.

[11] T. Wellhausen and A. Fiesser, "How to write a pattern? a rough guide for first-time pattern authors," in *16th European Conference on Pattern Languages of Programs*, Irsee, Germany, Jul. 2011. [Online]. Available: https://doi.org/10.1145/2396716.2396721

[12] S. Brinkmann and S. Kvale, *Doing Interviews*. Thousand Oaks, CA, USA: SAGE Publications Ltd, 2018.

[13] D. W. Turner III, "Qualitative interview design: A practical guide for novice investigators." *The Qualitative Report*, vol. 15, no. 3, pp. 754–760, Oct. 2022. [Online]. Available: https://doi.org/10.46743/2160-3715/2010.1178

[14] H. Montgomery *et al.*, "Hyperledger cactus whitepaper," Mar. 2022, (accessed Dic. 2023). [Online]. Available: https://github.com/hyperledger/cacti/blob/7bb39576080592919bea0ac89646b32105e174

[15] The Weaver Team, “Integration patterns,” (accessed Dic. 2023). [Online]. Available: https://labs.hyperledger.org/weaver-dlt-interoperability/docs/external/what-is-interoperability/integration-patterns

[16] Y. Liu, Q. Lu, H.-Y. Paik, and X. Xu, “Design patterns for blockchain-based self-sovereign identity,” in *Proceedings of the European Conference on Pattern Languages of Programs 2020*, ser. EuroPLoP ’20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3424771.3424802

[17] M. Wöhrer and U. Zdun, “Design patterns for smart contracts in the ethereum ecosystem,” in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1513–1520.