



**HAL**  
open science

# Verifiable and Friendly Metadata Management: Formal Blockchain to Property Graph Mapping

Anton Dolhopolov, Arnaud Castelltort, Anne Laurent

## ► To cite this version:

Anton Dolhopolov, Arnaud Castelltort, Anne Laurent. Verifiable and Friendly Metadata Management: Formal Blockchain to Property Graph Mapping. IDEAS 2024 - 28th International Database Engineered Applications Symposium, Aug 2024, Bayonne, France. hal-04677865

**HAL Id: hal-04677865**

**<https://hal.science/hal-04677865v1>**

Submitted on 30 Aug 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Verifiable and Friendly Metadata Management: Formal Blockchain to Property Graph Mapping

Anton Dolhopolov      Arnaud Castelltort      Anne Laurent

LIRMM, Univ. Montpellier, CNRS, Montpellier, France  
`firstname.lastname}@lirmm.fr`

**Abstract** Blockchain technology has gained popularity as a decentralized, tamper-proof, and verifiable data structure. Apart from financial transactions and assets management, there is also a growing interest in using it as a backbone for metadata management systems. Metadata systems aim to facilitate the organization of big data and to provide a comprehensive interface to users for data discovery and navigation. However, the fundamental design of the blockchain has the limitations in data querying and visualization that results in poor user experience. On the other side, the graph databases are widely adapted for building user friendly metadata systems with efficient relationship discovery capabilities. In our paper, we attempt to preserve the benefits of both types of systems through the integration of blockchain and property graph technologies. First, we formalize the structures used for metadata storage and a metadata conceptual model. Second, we provide the algorithms for mapping the (meta)data from the blockchain to the property graph model. And third, we show a prototype implementation of the integrated system to validate our proposal.

**Keywords:** Metadata Management   Blockchain   Property Graphs

## 1 Introduction

The proliferation of blockchain technology within enterprise information systems introduced a new direction of research. Beyond the financial bookkeeping applications (e.g. Bitcoin), there is a growing interest in applying this immutable, verifiable, and decentralized data structure for metadata management [5,6,8–10]. In this case, the metadata (e.g. format, location, size) is stored as the on-chain information about the off-chain data assets (e.g. datasets, media artifacts). Such architectural design aims to provide better scalability, verifiable records history, and to eliminate the downsides of storing the data in the blockchain directly [11].

Generally, metadata systems are required to provide an interface to the users for enabling data discovery, querying, analytics etc. In recent years, the graph databases become widely adapted for metadata management of big data platforms, both, in industrial tools [14,15] and in the academic works [7,12,19]. The main motivation for graph-model data storage is that it naturally supports data relationship modeling and it is not constrained to the strict schema enforcement compared to relational databases which enables scalability and flexibility [1].

A number of recent research studies have dedicated efforts to using graph technologies for enhancing blockchain capabilities. One of the directions is Directed Acyclic Graph (DAG) based ledgers [18]. But the goal of implementing the DAG structure is to overcome the fundamental transaction bandwidth issues seen in traditional blockchains.

Another direction focuses on conducting blockchain analytics by mapping the underlying on-chain data into the graphs. The majority of these analytical tasks are related to financial market predictions, fraud prevention, or community detection and evolution. According to the literature [13], almost all these systems are based on nodes representing account addresses or smart contracts, and edges representing asset flows or communication channels (e.g. contract calls).

We highlight that there is a growing demand for further research on the integration of graph technologies with the blockchain, particularly in the context of metadata management. As part of our contribution, we propose a formal model for metadata transformation from blockchain to property graphs and present a proof-of-concept system.

The paper is organized as follows: in Section 2 we review the related works and provide the necessary theoretical foundation. Section 3 depicts our formal model for generic and schema-dependent blockchain transformation. Section 4 outlines our prototype that implements the proposed formal models and in Section 5 we summarise our contribution and provide future research perspectives.

## 2 Background

In this section, we describe the related works of metadata systems and graph-enhanced blockchains. Then we introduce the blockchain and property graph formal models that will be used latter for our metadata mapping rules.

### 2.1 Related Works

Recently, there have been a growing number of proposals to use the blockchain structure as a decentralized metadata storage and management solution.

Garcia-Barriocanal et al. have researched the way of storing the metadata by using the globally accessible IPFS distributed technology [8]. Kumar et al. have considered the case of a widely utilized HDFS data storage and processing solution and proposed to improve the fault-tolerance mechanisms of the coordinator nodes by distributing the metadata information with the blockchain [9].

Demichev et al. have developed a system for tracking and managing the provenance metadata in the healthcare data-sharing scenarios [5], while Dolhopolov et al. have introduced a generalized blockchain-based metadata management system for a decentralized data mesh platform [6].

On the other side, as Sawadogo et Darmont point out [12], graph-based modelling is the most common way to implement the metadata systems thanks to the advantages of automatic information enrichment and advanced querying. Indeed, such commercial products as Apache Atlas [14] and DataHub Project [15] adopt

the property-graph data modelling for providing flexible and efficient metadata system. Eichler et al. have used a Neo4j graph database for implementing a general and expandable metadata model called HANDLE [7] and Ziegler et al. have used Neo4j for metadata management in the data lake context [19].

Unsurprisingly, the interest in enhancing the blockchain with graphs has been growing. In their vision paper, Bellomarini et al. have outlined the two-way positive impact cycle between the knowledge graphs (KG) and the blockchain [3]. On one side, KG can help with the knowledge-based smart contract generation process. On the other side, blockchain can be used for knowledge verification as it represents the unmodified source of data.

Cano-Benito et al. have proposed to use the blockchain for storing the ontologies in a way that would help to build the semantic web [4] and Fluree [16] database is an industrial solution based on Resource Description Framework technology that supports immutable data structure similar to the blockchain.

The closest work to ours is a contribution made by Tsoulas et al. [17]. The authors used the Neo4J database for directly storing the blockchain data as nodes and edges. They implemented the proof-of-work and proof-of-stake protocols with Python programming language and used the property-graph model to store the data. The work also verifies the solution against different real-world situations, like 51% Attack.

However, their proposal does not account for the integration of already running blockchain systems, but rather considers the situation of a new system bootstrap. It does not provide any discussion on the data migration from existing blockchains to the graph model what is in fact addressed in our proposal by the means of provided algorithms and mapping rules.

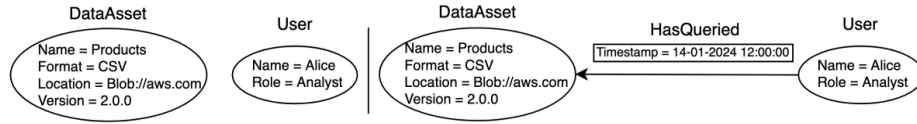
## 2.2 Blockchain Model

We describe a general model that can be applied to a wide range of blockchains, including Bitcoin, Ethereum, or Hypeledger Fabric that is used in our prototype. Due to the space limits, we omitted a complete formal definition and rather provide a general description thereafter.

Blockchain  $\mathcal{B}$  is a sequence of cryptographically signed blocks. The blocks are made of transactions issued by user accounts and a header  $h$  that has a hash value of all transactions within that block. The first block in the blockchain is a genesis block that may contain arbitrary data. Each following block contains a header that also has a reference to the previous header and its hash value (established via function  $ref_h$ ). This way the blockchain provides a verifiable and tamper-proof data structure.

Usually, transactions change the account state. They are comprised of the operations over the assets, e.g. substitution of some currency from account  $x$  and equivalent amount addition to the account  $y$ . We can write it down as a *key value* pair  $(k, v)$  that represents the account label and balance respectively.

In scenario of using the blockchain as a metadata system, we can apply this approach for storing information about the data assets by using  $(k, v)$  pairs for describing its metadata entries (e.g.  $k = format$  and  $v = CSV$ ).



**Fig 1** Property graph example: node types (left) and edge formation (right)

Depending on the blockchain implementation, user transaction  $tx$  can also contain different platform-related information. In our model, we assume that each transaction is composed of a user signature (for establishing its ownership) and a set of properties  $P$  reflecting the metadata, that is  $tx = (sign(a), P)$ .

Usually, each transaction is signed by a single account which establishes its ownership rights. But sometimes, it is possible to have a *multi signature* transaction, which requires the participation of more than a single account to consider a transaction to be valid. In this case, we will use a special function  $\mu_{ms}$  to denote an “ownership” of the transaction  $tx_i$  across multiple accounts.

Attention should be paid to the header reference function  $ref_h$  that always holds true, except the case of the genesis block (absence of the block before). To overcome it, we define a self-referencing function  $ref_h(h_0) = h_0$ .

### 2.3 Property Graph Model

In recent years property graphs (PGs) have gained popularity as a NoSQL data model. It provides a way to efficiently represent the relationships and connections of natural phenomena in the form of nodes and edges that are enriched with properties. For instance, data assets and users can be seen as nodes while the actions performed by users over the data will be represented as edges. When a user queries an asset, we create an edge with a label *HasQueried* from the user to the asset of interest and add the timestamp when it was done as an edge property. In Figure 1 we show the described property graph example.

We define the formal PG model as a tuple  $\mathcal{PG} = (N, E, L, P, lbl, edge, prop)$ :

- $N$  is a finite set of **nodes**,  $E$  is a finite set of **edges**,  $L$  is a finite set of **labels**, and  $P$  is a finite set of **properties** such that  $N \cap E \cap P \cap L = \emptyset$
- $lbl : N \cup E \rightarrow L$  is a function that assigns a single label to a node or an edge
- $edge : E \rightarrow N \times N$  is a function that assigns each edge to a pair of nodes  $(n, n')$ , where  $n$  is the source node and  $n'$  is the target node
- $prop : N \cup E \rightarrow \mathcal{P}^+(P)$  is a partial function that associates a node or an edge with non-empty set of properties  $P$ , such that  $p = (k, v)$ ,  $p \in P$ ,  $k \in K$ ,  $v \in V$ .

As follows,  $K$  is a set of possible property keys and  $V$  is a set of possible property values. In Figure 1 the labels *User* and *DataAsset* from set  $L$  are also assigned to nodes and offer further functionality for distinguishing node types.

In Section 1 we pointed out that often metadata systems are implemented with a property graph model [12]. The flexible and schema-free nature of the PG

model supports the quick evolution of enterprise data assets and their relationships. Modern graph databases do not require any predefined data schema for using it (contrary to relational databases). Moreover, there is always an option to enforce the user-provided schema to guarantee data consistency and integrity.

In general, labels are used for defining the semantic meaning of the data through the user schema. In Figure 1 *HasQueried* label describes the nature of the action relationship between the two nodes. At all times, we assume to have a PG schema that defines the structure of possible relationships between the nodes. Put it formally, schema  $\mathcal{S}_{PG}$  is composed of node labels  $L_N$  and edge labels  $L_E$ , such that  $L_N \cup L_E = L$ ,  $L_N \cap L_E = \emptyset$ ; and a function  $\gamma : L_N \times L_N \rightarrow L_E$  that defines allowed relationships between a given pair of nodes.

In this section, we have described the related works and the necessary theoretical background. The following section goes more into the detail of metadata mapping from the blockchain to PG data structure.

### 3 Designing a Formal Model for Metadata Mapping

In this section, we outline the first part of our contribution. We start by describing the properties of the data transformation we want to comply with. Then we consider a running example of the metadata system and provide a generic, schema-independent conversion. As a subsequent step, we propose a conceptual metadata model and introduce a procedure for schema-dependent conversion.

Due to the space limits, we note that the complete formal rules of the mapping are provided as part of the prototype code that is discussed in Section 4.

#### 3.1 Properties of Metadata Mapping

We recall three properties of the database transformation from Angles et al. [2], which are: computability, information and semantic preservation. We briefly introduce it here and we address the reader to the original paper for more details.

*Computability* indicates that for any two given databases  $\mathcal{D}_1$  and  $\mathcal{D}_2$  there exists an algorithm that computes a database mapping  $\mathcal{DM} : \mathcal{D}_1 \rightarrow \mathcal{D}_2$ . Our goal is to provide an algorithm showing that mapping  $\mathcal{DM} : \mathcal{B} \rightarrow \mathcal{PG}$  is feasible.

*Information preservation* means that for a computable mapping  $\mathcal{DM}$  there exists an inverse translation  $\mathcal{DM}^{-1} : \mathcal{D}_2 \rightarrow \mathcal{D}_1$ , or that  $\mathcal{D} = \mathcal{DM}^{-1}(\mathcal{DM}(\mathcal{D}))$ . It is a required property for us since we are only interested in translations that don't lose the source information.

*Semantic preservation* indicates that the result of the mapping is a **valid** database. In this context, a valid database means a database instance  $\mathcal{I}$  which is compliant with rules and constraints of the target schema  $\mathcal{S}$ , denoted as  $\mathcal{I} \models \mathcal{S}$ . To be more precise, when we discuss database mapping, we assume that any database is composed of an instance and a schema, or that  $\mathcal{D} = (\mathcal{I}, \mathcal{S})$ . If the schema is not defined, it means that  $\mathcal{D} = (\mathcal{I}, \emptyset)$ , so we will use the terms "database" and "instance" interchangeably.

### 3.2 Generic Transformation from Blockchain to Property Graph

As a first step, we consider an existing metadata system based on Hyperledger Fabric [6]. Our first goal is to show the feasibility of transforming the on-chain metadata from the blockchain to the property graph structure.

A running example of the blockchain-based catalog, presented in Figure 2, has three blocks modeling different metadata entries. The first block is a genesis block with a self-referencing header. It describes a data asset in transaction  $TxA1$  and its owner (displayed to the left). The second block has a similar structure, where the header points to the previous block as expected. In our case, block  $TxB1$  has two signing accounts and one transaction that references another one:  $TxB1 \rightarrow TxA1$ , which is valid according to our formal model. Finally, the third block describes 2 data assets from different users.

To perform a generic, schema-independent metadata mapping, we introduce a generic property graph schema  $\mathcal{S}_G$  that defines the structure of the generated graph. It enables us to map any blockchain-based catalog into the PG-based catalog. Our target schema consists of a set of node labels  $L_N$  and a set of edge labels  $L_E$ , where  $L_N = \{UserAccount, BlockHeader, Transaction\}$ ,  $L_E = \{PreviousHeader, ReferencesTo, BelongsTo, Signed\}$ ,  $L_N \cup L_E = L_G$ .

---

#### Algorithm 1: Generic Metadata Mapping

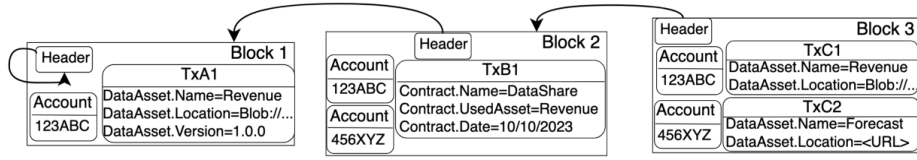
---

```

Data Sequence of blocks B
Result Nodes N & edges E
1  N =  $\emptyset$ , E =  $\emptyset$ ;
2  for b  $\in$  Sorted B do
3    h = b.header;
4    lbl(h) = BlockHeader;
5    Push N, h) /* appending the header h to the set of nodes N */;
6    e =  $\emptyset$ ;
7    lbl(e) = PreviousHeader, edge(e) = (h, ref(h));
8    Push E, e);
9    for tx  $\in$  b.transactions do
10   n =  $\emptyset$ , e =  $\emptyset$ , a =  $\emptyset$ ;
11   lbl(n) = Transaction, n.P = tx.P;
12   Push N, n);
13   lbl(e) = BelongsTo, edge(e) = (n, h);
14   Push E, e);
15   if  $\mu(tx)$  is not in N then
16     | lbl(a) = UserAccount, a.Sign = tx.sign;
17     | Push N, a);
18   else
19     | a = Get N, tx.sign) /* finding a user a by the transaction signature in N */
20   end
21   e =  $\emptyset$ ;
22   lbl(e) = Signed, edge(e) = (a, n);
23   Push E, e);
24   for r  $\in$  ref(tx) do
25     | e =  $\emptyset$ ;
26     | lbl(e) = ReferencesTo, edge(e) = (n, r);
27     | Push E, e);
28   end
29 end
3 end

```

---



**Fig 2** A running example of the blockchain-based metadata catalog.

We provide an Algorithm 1 that guarantees the computability of the mapping  $\mathcal{DM}_G : \mathcal{B} \rightarrow \mathcal{PG}$ . To prove that  $\mathcal{DM}_G$  is information preserving we need to have a mapping  $\mathcal{DM}_G^{-1}$  such that  $\mathcal{I} = \mathcal{DM}_G^{-1}(\mathcal{DM}_G(\mathcal{I}))$  for any given instance  $\mathcal{I}$  that represents blockchain catalog  $\mathcal{B}$ . It should be noted that we consider a blockchain model as defined in Section 2.2. Therefore, it is sufficient to recover the initial database structure, including chain blocks, headers, transactions, accounts, and their relations. The solution to this is provided in the Algorithm 2.

The output of the generic transformation  $\mathcal{DM}_G$  is a graph  $\mathcal{PG} = (\mathcal{I}_G, \mathcal{S}_G)$ , where  $\mathcal{I}_G$  is a graph database instance as described in Section 2.3 and  $\mathcal{S}_G$  is a generic schema created for any input blockchain catalog. It means that  $\mathcal{I}_G \models \mathcal{S}_G$  by definition and that transformation  $\mathcal{DM}_G$  is semantic preserving.

The result of the proposed schema-independent transformation complies with all three properties: it is computable, information and semantic preserving. One of the benefits of this mapping is that metadata stored in the form of property-graph model gives a way to detect transaction clusters or to query all transactions created by a given user by simply following the outgoing *Signed* edges.

However, most of the time the metadata system users will define their own catalog schema which is enforced within the blockchain. Therefore, we have to have a schema-dependent mapping in order to generate an appropriate graph.

---

### Algorithm 2: Inverse Metadata Mapping

---

```

Data Nodes N & edges E
Result Sequence of blocks B
1  $B \leftarrow \emptyset$ ;
2  $H \leftarrow \text{Find } BlockHeader$  /* nding all nodes with BlockHeader node label */;
3 for  $h \in \text{Sorted } H$  do
4    $b \leftarrow \emptyset$ ;
5    $b.header \leftarrow h$ ;
6    $h \leftarrow \text{Find } PreviousHeader, h$  /*get a node by traversing a PreviousHeader edge*/;
7    $ref(b.header) \leftarrow h$ ;
8    $b.transactions \leftarrow \emptyset$ ;
9    $T \leftarrow \text{Find } BelongsTo, h$ ;
10  for  $n \in T$  do
11     $a \leftarrow \text{Find } Signed, n$ ;
12     $tx \leftarrow (a, n.P)$ ;
13    for  $r \in \text{Find } ReferencesTo, n$  do
14       $\text{Push } tx.references, r$ 
15    end
16     $\text{Push } b.transactions, tx$ 
17  end
18 end

```

---



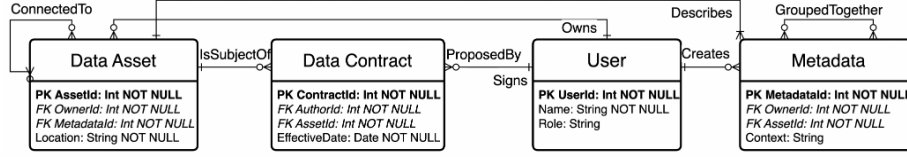


Fig 3 Conceptual model of metadata catalog.

### 3.3 Conceptual Metadata Catalog Model

Before implementing the schema-dependent metadata mapping, we need a conceptual catalog model. Considering the vast amount of different metadata models, we adopt a simple, generic and extensible model called HANDLE [7].

We extend the HANDLE according to our previous running example and present an Entity-Relationship diagram model in Figure 3. We omit a full definition here, but briefly state that it is seen as a tuple  $\mathcal{M} = (D, M, C, U)$ , where:  $D$  is a finite set of **data** assets,  $M$  is a finite set of assets **metadata**,  $C$  is a finite set of data **contracts**,  $U$  is a finite set of system **users**.

In this model, a data asset represents the raw data and it has links to the storage location, connected assets, its metadata, and owner. Metadata describes a single asset on a higher abstraction, like when, who, and how it was created. There can be many metadata entries related to the same asset but with a different context (creation, access, etc). A data contract defines a data-sharing relation between different parties, or simply users. It also has two “ends” of a relation: the contract should be proposed by a single user. At the same time, users can sign zero or many data contracts.

Eventually, we want to check if a given blockchain catalog  $\mathcal{B} = (\mathcal{I}_{\mathcal{M}}, \mathcal{S}_{\mathcal{M}})$  is valid w.r.t. the model  $\mathcal{M}$ , or that  $\mathcal{I}_{\mathcal{M}} \models \mathcal{S}_{\mathcal{M}}$ . Since we can’t have explicit labels of the references in the blockchain, we fall back to considering only transaction types and the implicit link map between them. We assume that  $L_{\mathcal{M}}$  is a set of entity type labels corresponding to model  $\mathcal{M}$  and that  $\lambda : tx \rightarrow L_{\mathcal{M}}$  is a function that returns the label of a given transaction. We say that  $\mathcal{I}_{\mathcal{M}} \models \mathcal{S}_{\mathcal{M}}$ , if :

- **M1:** for any  $tx \in T$ , such that  $\lambda(tx) = DataAsset$ , there is a  $tx' \in T$ , where:  
 $\lambda(tx') = Metadata \quad ref(tx') = tx \quad \mu(tx) = \mu(tx')$ ;
- **M2:** for any  $tx \in T$ , such that  $\lambda(tx) = Metadata$ , there is a  $tx' \in T$ , where:  
 $\lambda(tx') = DataAsset \quad ref(tx) = tx'$ ;
- **M3:** for any  $tx \in T$ , where  $\lambda(tx) = DataContract$ , there is  $tx' \in T$ , where:  
 $\lambda(tx') = DataAsset \quad ref(tx) = tx' \quad \mu(tx) \neq \mu(tx')$ ;
- **M4:** for any pair  $(tx, tx') \in T$ , such that:  
 $\lambda(tx) = \lambda(tx') = DataAsset \quad ref(tx) = tx' \quad \mu(tx) \neq \mu(tx')$   
 $\triangleright$  there exists a transaction  $\hat{tx}$ , such that:  
 $\lambda(\hat{tx}) = DataContract \quad \mu_{ms}(\hat{tx}) = (\mu(tx), \mu(tx'))$ ;

where  $L_{\mathcal{M}} = \{DataAsset, Metadata, DataContract\}$ . We intentionally omit the distinction of a *User* entity because this information is always present as part of the considered transaction.

### 3.4 Schema Dependent Metadata Catalog Mapping

As a last part of our formalisation, we design a schema-dependent catalog mapping  $\mathcal{DM}_S : \mathcal{B}_M \rightarrow \mathcal{PG}_M$ , where  $\mathcal{B}_M = (\mathcal{I}, \mathcal{S})$  and  $\mathcal{PG}_M = (\mathcal{IPG}, \mathcal{SPG})$  are valid databases w.r.t. previously defined model  $\mathcal{M}$ .

---

#### Algorithm 3: Schema-Dependent Metadata Mapping

---

```

Data Sequence of blocks B
Result Nodes N & edges E
1  N =  $\emptyset$ , E =  $\emptyset$ ;
2  for  $b_i \in \text{Sorted } B$  do
3    | h = b.header, lbl(h) = Block, h.OrderNo = i;
4    | Push N, h) /* appending the header h to the set of nodes N */;
5    | e =  $\emptyset$ , lbl(e) = Previous, edge(e) = (h, ref(h));
6    | Push E, e) /* appending the edge e to the set of edges E */;
7    | for tx  $\in$  b.transactions do
8      | n =  $\emptyset$ , e =  $\emptyset$ , a =  $\emptyset$ ;
9      | lbl(n) = (tx, n.P = tx.P;
10     | Push N, n);
11     | lbl(e) = BelongsTo, edge(e) = (n, h);
12     | Push E, e);
13     | if  $\mu(tx)$  is not in N then
14       | | lbl(a) = User, a.Sign = tx.sign;
15       | | Push N, a);
16     | else
17       | | a = Get N, tx.sign) /* finding a user a by the transaction signature in N */
18     | end
19     | for r  $\in$  ref(tx) do
20       | e =  $\emptyset$ , e =  $\emptyset$ ;
21       | edge(e) = (n, Get N, r);
22       | if lbl(n) is DataAsset then
23         | | lbl(e) = ConnectedTo;
24         | | lbl(e) = Owns, edge(e) = (n, a);
25         | | Push E, e, e );
26       | else
27         | if lbl(n) is Metadata then
28           | | if (r) is DataAsset then
29             | | | lbl(e) = Describes;
30           | | else
31             | | | lbl(e) = GroupedTogether;
32           | | end
33           | | lbl(e) = Creates, edge(e) = (n, a);
34           | | Push E, e, e );
35         | else
36           | /* Means that lbl(n) is DataContract */;
37           | lbl(e) = IsSubjectOf;
38           | (a1, a2) =  $\mu_{ms}(tx)$ ;
39           | if a2 is  $\emptyset$  then
40             | | lbl(e) = ProposedBy, edge(e) = (n, a);
41             | | Push E, e, e );
42           | else
43             | | lbl(e) = ProposedBy, edge(e) = (n, a );
44             | | lbl(e2) = Signs, edge(e2) = (n, a2);
45             | | Push E, e, e1, e2);
46           | end
47         | end
48       | end
49     | end
50   | end
51 end

```

---

The outlined catalog conversion  $\mathcal{DM}_S : \mathcal{B}_M \rightarrow \mathcal{PG}_M$  is computable and we provide a corresponding solution in the Algorithm 3.

The obtained property graph  $\mathcal{PG}_M$  is a valid database, meaning that  $\mathcal{DM}_S$  is semantic preserving. We note that resulting schema  $\mathcal{S}_{PG}$ , as well as  $\mathcal{S}$ , is directly derived from the conceptual model  $\mathcal{M}$ . It is composed of node labels  $L_N = \{DataAsset, Metadata, User, DataContract\}$  and edge labels  $L_E = \{Describes, Owns, Creates, ProposedBy, Signs, IsSubjectOf, ConnectedTo, GropedTogether\}$ . The respective entities are guaranteed by node generation code lines 8-10 and required relationships are guaranteed by edge generation code lines 20-44.

To guarantee the information preservation of our mapping based on model  $\mathcal{M}$ , we follow a similar approach as in generic algorithm and therefore introduce the block ordering with code lines 3-6 and 11-12.

In this section, we proposed two formal procedures for performing metadata catalog mapping from blockchain to property graph database. The schema-independent procedure allows us to obtain a generic graph that enhances the user’s capacity to traverse and analyse the metadata. At the same time, the schema-dependent procedure also provides a way to systematize the metadata according to the user-defined conceptual model. In addition, both translations are computable, information and semantic preserving.

In the following section we show a proof-of-concept implementation of our formal models alongside its performance assessment.

## 4 Implementing Metadata Mapping from Blockchain to Property Graph

This section provides details about the developed system used for validating the defined formal models. Then it continues with evaluation methodology, provides quantitative and qualitative results, and concludes with a results discussion.

### 4.1 Implementation Details

For validating our theoretical proposal with a practical evidence, we have developed a proof-of-concept (PoC) system for testing the transformation of metadata information from blockchain to property graph model from two perspectives.

First aspect includes the validation of the real-world, end-to-end transformation or mapping system, which provides a suitable interface to the users. Our PoC has a decentralized ledger deployed with Hyperledger Fabric (HLF) and a Neo4J graph database that supports the property graph model. The mapping module is implemented as a middleware RESTful API service with a Swagger support. In addition to metadata transformation (Mapping Module), this middleware service also enables the communication channels with HLF (via LedgerDriver) and Neo4J (via GraphDriver) with direct HTTP(s) requests or a graphical interface provided by Swagger. The architecture of the system is presented in Figure 4.

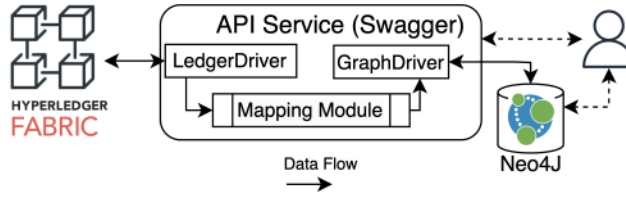


Fig 4 Prototype system architecture.

We use the HLF because it naturally extends our running example from Section 3.2 and enables us to store the information as key-document or key-value pairs which are required by the formal models. We employ the Neo4J database because it is a leading, ACID-compliant graph storage and processing solution that has a lot of driver libraries.

Second aspect includes testing the performance and scalability of the proposed metadata catalog mapping. We developed two algorithms for generic and schema-dependent mappings based on the formal models in Section 3.

For excluding the external factors on the algorithm performance evaluation, such as database indexing or network delays, we use beforehand extracted ledger metadata information which is stored according to the blockchain model from Section 2. The output of each transformation algorithm is a text file containing the instructions necessary for creating required nodes, edges, and their corresponding labels and properties. The instructions are written in Cypher querying language that is supported by Neo4J database. Both algorithms support the parallel computation model that is used for scalability assessment.

The system’s code and full formal rules are available on Google Drive <sup>1</sup>.

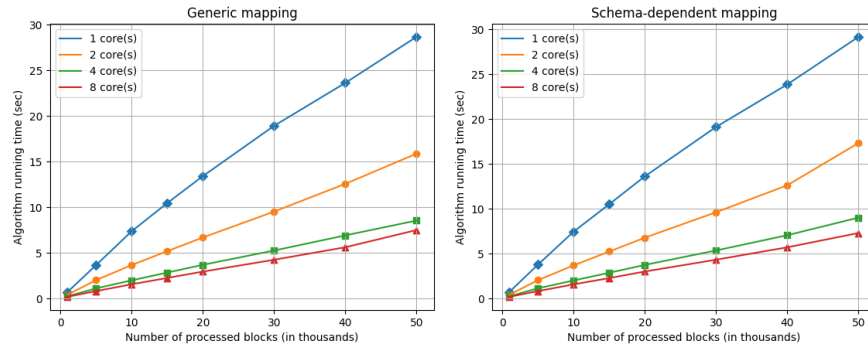
## 4.2 Methodology and Experimental Setup

In order to conduct the prototype performance assessment we used the stable release of Ubuntu 22.04 (desktop version) with default post-installation parameters. The system hardware characteristics were: AMD Ryzen 5600x CPU with 6 cores (12 threads), 32 GB of RAM, 1 TB of HDD disk storage with 5400 rpm. The transformation module used in performance benchmark was implemented as a Go binary program using version 1.20. The provided input files were represented as a JSON files and the output text files contained Cypher-based instructions.

Our primary metric of the benchmark was the measurement of a transformation time from the underlying files representing metadata in blockchain’s format to the property graph nodes and edges. We employed standard library *time* package<sup>2</sup> to test the mapping duration with ms precision. We run the tests 10 times per transformation type per varying number of blocks.

<sup>1</sup> <https://drive.google.com/le/d/1ueml6ZT5UF0qoLumUB-KVqhZW0yFq5zT>

<sup>2</sup> <https://pkg.go.dev/time>



**Fig 5** Algorithm running time: generic (left) and schema-dependent method (right).

To conduct the scalability evaluation, we tested our application with variable number of utilized CPU cores. To achieve it on the same testing hardware, we used the *runtime.GOMAXPROCS* package function<sup>3</sup> that provides a way to specify the maximum number of utilized processor cores by a program. We used randomly generated data to populate the Hyperledger Fabric ledger and then extracted it into the JSON.

### 4.3 Results and Discussions

The transformation running time results are presented in Figure 5. It reveals that our algorithms exhibit acceptable performance and scalability properties. For instance, processing 10 thousands blocks (approximately 220 Mb) with 1 processor core takes 7.5 seconds, while using 4 cores reduces the processing time to 2 seconds, on average, with the generic and the schema-dependent methods. If we scale up the ledger to a larger number of blocks, we observe that the processing time increases approximately linearly for both methods. This trend is evident when handling the 50 thousands blocks (approximately 1100 Mb), where the execution runtime, on average, extends to about 29 seconds for 1 core setup and to about 9 seconds for 4 core setup. The proposed transformation algorithm passes through source blocks and transactions only once, meaning that it has the complexity of  $O(n)$ . This is also evident from the performance evaluation since the processing time grows linearly to the input data size.

We assume that the low improvement between 4 and 8 cores setups is due to the hardware constraint, which has only 6 full cores with hyper-threading.

The visual difference between the standard ledger visualization and graph view is presented in Figure 6. On the left side of the figure, we can see transaction information in the form of JSON properties. The tool, called Hyperledger Explorer, automatically extracts the ledger data and stores it in the relational database. In fact, we see that such metadata presentation provides poor usability and leaves the problem of navigation and querying to the user.

<sup>3</sup> <https://pkg.go.dev/runtime#GOMAXPROCS>

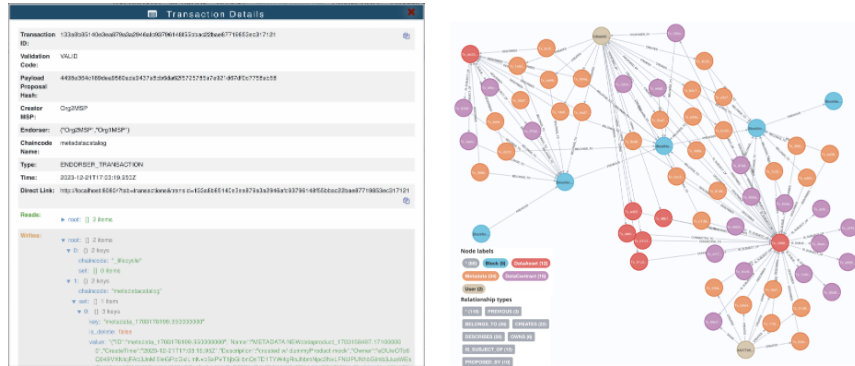


Fig 6 Explorer transaction view (left) and Neo4J web UI sub-graph view (right).

On the opposite side, the built-in web interface to the graph data, provided by the Neo4J, models a lot of different entities and relationships on a single screen, which has considerably higher utility compared to the previous tool. Additionally, users can employ a Cypher querying language for efficient data querying, manipulation, and discovery. Cypher syntax is similar to SQL and it provides a way to easily define the querying graph patterns naturally.

## 5 Conclusions and Further Research

To conclude, this paper presents a novel way for improving data discovery and querying of the decentralized metadata management system built on top of the blockchain. It discusses the research on the integration of blockchains with graphs and highlights the missing functionalities that are satisfied by our method.

The proposed approach is based on two parts. First, we define the formal models for generic (schema-independent) and schema-dependent metadata transformation from the ledger storage to the property graph database. Furthermore, the models comply with three essential properties of database model mapping: computability, information and semantic preservation.

Second, we design and implement a prototype system that proves the feasibility and correctness of the formal models as and we provide the transformation algorithm performance and scalability evaluation. The system is based on a widely deployed permissioned blockchain - Hyperledger Fabric, and a leading transactional graph database - Neo4J.

We envision further research on several axes. We note that even though the currently used metadata model is generic and extensible, the users may still need to apply the proprietary metadata models. To achieve it, we are required to have a comprehensive metadata transformation module that processes any user-defined schema. Moreover, such a module will also benefit from incorporating the processing of a time dimension, that is from supporting the evolution of properties of entities and relationships, which is lacking as of today.

## References

1. Angles, R., Gutierrez, C.: An introduction to graph data management. *Graph Data Management: Fundamental Issues and Recent Developments* pp. 1–32 (2018)
2. Angles, R., Thakkar, H., Tomaszuk, D.: Mapping rdf databases to property graph databases. *IEEE Access* **8**, 86091–86110 (2020)
3. Bellomarini, L., Nissl, M., Sallinger, E.: Blockchains as knowledge graphs-blockchains for knowledge graphs(vision paper). In: *KR4L@ECAI*. pp. 43–51 (2020), <https://api.semanticscholar.org/CorpusID:244715730>
4. Cano-Benito, J., Cimmino, A., García-Castro, R.: Towards blockchain and semantic web. In: *Business Information Systems Workshops: BIS 2019 International Workshops*, Seville, Spain, June 26–28, 2019, Revised Papers 22. pp. 220–231. Springer (2019)
5. Demichev, A., Kryukov, A., Prikhodko, N.: The approach to managing provenance metadata and data access rights in distributed storage using the hyperledger blockchain platform. In: *Ivannikov Ispras Open Conference*. IEEE (2018)
6. Dolhopolov, A., Castelltort, A., Laurent, A.: Implementing a blockchain-powered metadata catalog in data mesh architecture. In: *International Congress on Blockchain and Applications*. pp. 348–360. Springer (2023)
7. Eichler, R., Giebler, C., Gröger, C., Schwarz, H., Mitschang, B.: Modeling metadata in data lakes—a generic model. *Data & Knowledge Engineering* (2021)
8. García-Barriocanal, E., Sánchez-Alonso, S., Sicilia, M.A.: Deploying metadata on blockchain technologies. In: *Metadata and Semantic Research: 11th International Conference, MTSR 2017, Tallinn, Estonia, November 28–December 1, 2017, Proceedings 11*. pp. 38–49. Springer (2017)
9. Kumar, D.S., Rahman, M.A.: Simplified hdfs architecture with blockchain distribution of metadata. *International Journal of Applied Engineering Research* (2017)
10. Liu, L., Li, X., Au, M.H., Fan, Z., Meng, X.: Metadata privacy preservation for blockchain-based healthcare systems. In: *Database Systems for Advanced Applications: 27th International Conference, DASFAA 2022, Virtual Event, April 11–14, 2022, Proceedings, Part I*. pp. 404–412. Springer (2022)
11. Paik, H.Y., Xu, X., Bandara, H.D., Lee, S.U., Lo, S.K.: Analysis of data management in blockchain-based systems: From architecture to governance. *IEEE Access* **7**, 186091–186107 (2019)
12. Sawadogo, P., Darmont, J.: On data lake architectures and metadata management. *Journal of Intelligent Information Systems* **56**(1), 97–120 (2021)
13. Song, J., Zhang, P., Qu, Q., Bai, Y., Gu, Y., Yu, G.: Why blockchain needs graph: a survey on studies, scenarios, and solutions. *Journal of Parallel and Distributed Computing* p. 104730 (2023)
14. Apache Foundation: Atlas. <https://atlas.apache.org>, accessed: 14.01.2024
15. DataHub Project: The metadata platform for the modern data stack. <https://datahubproject.io/>, accessed: 14.01.2024
16. Fluree PBC: Flureedb. <https://developers.fluree.com/>, accessed: 14.01.2024
17. Tsoulas, K., Palaiokrassas, G., Fragkos, G., Litke, A., Varvarigou, T.A.: A graph model based blockchain implementation for increasing performance and security in decentralized ledger systems. *IEEE Access* **8**, 130952–130965 (2020)
18. Wang, Q., Yu, J., Chen, S., Xiang, Y.: Sok: Diving into dag-based blockchain systems. *arXiv preprint arXiv:2012.06128* (2020)
19. Ziegler, J., Reimann, P., Keller, F., Mitschang, B.: A graph-based approach to manage cae data in a data lake. *Procedia CIRP* **93**, 496–501 (2020)

## A Generic Mapping Rules.

We define the generic metadata catalog mapping  $\mathcal{DM}_G$  as follows:

- **G1:** for any account  $a \in A$ :
  - ▷ there will be a node  $n \in N$ , where  $lbl(n) = UserAccount$ ;
- **G2:** for any header  $h \in H$ :
  - ▷ there will be a node  $n \in N$ , where  $lbl(n) = BlockHeader$ ;
- **G3:** for any transaction  $tx \in T$ :
  - ▷ there will be a node  $n \in N$ , such that:
 
$$lbl(n) = Transaction \quad prop(n) = \pi(tx);$$
- **G4:** for any pair of headers  $(h, h') \in H$ , where  $ref(h) = h'$ :
  - ▷ there will be an edge  $e \in E$ ,  $edge(e) = (n, n')$ , such that:
 
$$lbl(e) = PreviousHeader \quad lbl(n) = lbl(n') = BlockHeader$$
 where nodes  $(n, n')$  correspond to block headers  $(h, h')$  respectively;
- **G5:** for any pair  $(tx, tx') \in T$ , where  $tx' \in ref(tx)$ :
  - ▷ there will be an edge  $e \in E$ ,  $edge(e) = (n, n')$ , such that:
 
$$lbl(e) = ReferencesTo \quad lbl(n) = lbl(n') = Transaction$$
 where nodes  $(n, n')$  correspond to transactions  $(tx, tx')$  respectively;
- **G6:** for any pair  $(h, tx)$  that belongs to the same block  $b \in B$ :
  - ▷ there will be an edge  $e \in E$ ,  $edge(e) = (n, n')$ , such that :
 
$$lbl(n) = Transaction \quad lbl(e) = BelongsTo \quad lbl(n') = BlockHeader;$$
- **G7:** for any account  $a \in A$  and  $tx \in T$ , where  $sign(a) \in tx$ , there will be:
  - ▷ an edge  $e \in E$ ,  $edge(e) = (n, n')$ , such that :
 
$$lbl(n) = UserAccount \quad lbl(e) = Signed \quad lbl(n') = Transaction.$$

## B Schema-Dependent Mapping Rules.

We define the schema-dependent mapping  $\mathcal{DM}_S$  as follows:

- **S1:** for any  $tx \in T$ , such that  $\lambda(tx) = DataAsset$ , there will be:
  - ▷ a node  $n \in N$ , where  $lbl(n) = DataAsset$ ;
- **S2:** for any  $tx \in T$ , such that  $\lambda(tx) = Metadata$ , there will be:
  - ▷ a node  $n \in N$ , where  $lbl(n) = Metadata$ ;
- **S3:** for any  $tx \in T$ , such that  $\lambda(tx) = DataContract$ , there will be:
  - ▷ a node  $n \in N$ , where  $lbl(n) = DataContract$ ;
- **S4:** for any property  $p \in \pi(tx)$ , where  $\lambda(tx) \in L_M$  there will be:
  - ▷ a corresponding property  $p' \in prop(n)$ , where node  $n$  corresponds to  $tx$ ;
- **S5:** for any pair of transactions  $(tx, tx') \in T$ , such that:
 
$$\lambda(tx) = DataAsset \quad \lambda(tx') = Metadata \quad ref(tx') = tx$$
  - ▷ **S5.1:** there will be a node  $n^* \in N$ , such that  $lbl(n^*) = User$
  - ▷ **S5.2:** there will be an edge  $e_1 \in E$ ,  $edge(e_1) = (n', n)$ , such that:
 
$$lbl(n') = Metadata \quad lbl(e_1) = Describes \quad lbl(n) = DataAsset$$
  - ▷ **S5.3:** there will be an edge  $e_2 \in E$ ,  $edge(e_2) = (n^*, n')$ , such that:
 
$$lbl(n^*) = User \quad lbl(e_2) = Creates \quad lbl(n') = Metadata$$



- ▷ **S5.4:** if  $\mu(tx) = \mu(tx')$ , there will be  $e_3 \in E$ ,  $edge(e_3) = (n^*, n)$ , where:  
 $lbl(n^*) = User \quad lbl(e_3) = Owns \quad lbl(n) = DataAsset$
- **S6:** for any pair of transactions  $(tx, tx') \in T$ , such that:  
 $\lambda(tx) = DataAsset \quad \lambda(tx') = DataContract \quad ref(tx') = tx$ 
  - ▷ **S6.1:** there will be a node  $n^* \in N$ , where  $lbl(n^*) = User$
  - ▷ **S6.2:** there will be an edge  $e_1 \in E$ ,  $edge(e_1) = (n', n^*)$ , such that:  
 $lbl(n') = DataContract \quad lbl(e_1) = ProposedBy \quad lbl(n^*) = User$
  - ▷ **S6.3:** and there will be an edge  $e_2 \in E$ ,  $edge(e_2) = (n, n')$ , such that:  
 $lbl(n) = DataAsset \quad lbl(e_2) = IsSubjectOf \quad lbl(n') = DataContract$
- **S7:** for any pair of transactions  $(tx, tx') \in T$ , such that:  
 $\lambda(tx) = \lambda(tx') = DataAsset \quad tx \in ref(tx')$ 
  - ▷ **S7.1:** there will be an edge  $e \in E$ ,  $edge(e) = (n', n)$ , such that:  
 $lbl(n) = lbl(n') = DataAsset \quad lbl(e) = ConnectedTo$
  - if  $\mu(tx) \neq \mu(tx')$ , then there will be:
    - ▷ **S7.2.1** three nodes  $(n_1, n_2, n_3) \in N$ , such that:  
 $lbl(n_1) = lbl(n_2) = User \quad lbl(n_3) = DataContract$   
where  $\mu(tx)$  correspond to  $n_1$  and  $\mu(tx')$  to  $n_2$
    - ▷ **S7.2.2** and three edges  $(e_1, e_2, e_3) \in E$ , such that:  
 $edge(e_1) = (n, n_3) \quad edge(e_2) = (n_3, n_1) \quad edge(e_3) = (n_2, n_3)$   
 $lbl(n) = DataAsset \quad lbl(e_1) = IsSubjectOf \quad lbl(n_3) = DataContract$   
 $lbl(n_2) = User \quad lbl(e_3) = Signs \quad lbl(n_3) = DataContract$   
 $lbl(n_3) = DataContract \quad lbl(e_2) = ProposedBy \quad lbl(n_1) = User$
- **S8:** for any pair of transactions  $(tx, tx') \in T$ , such that:  
 $\lambda(tx) = \lambda(tx') = Metadata \quad tx \in ref(tx')$ 
  - ▷ there will be an edge  $e \in E$ ,  $edge(e) = (n, n')$ , such that:  
 $lbl(n) = lbl(n') = Metadata \quad lbl(e) = GropedTogether$
where  $(n, n')$  correspond to the  $(tx, tx')$  respectively;
- **S9:** for any pair  $(h_i, tx)$  of a header and transaction, and block  $b_i$ , such that:  
 $h_i \in b_i \quad tx \in b_i \quad b_i \in B \quad \lambda(tx) \in L_{\mathcal{M}}$ 
  - ▷ there will be two nodes  $(n, n') \in N$ , such that:  
 $lbl(n) = \lambda(tx) \quad lbl(n') = Block \quad prop(n').OrderNo = i$
  - ▷ there will be edge  $e \in E$ ,  $edge(e) = (n, n')$ , such that:  
 $lbl(n) \in L_N \quad lbl(e) = BelongsTo \quad lbl(n') = Block;$
  - ▷ and there will be edge  $e' \in E$ ,  $edge(e') = (n', n'')$ , such that:  
 $lbl(n') = lbl(n'') = Block \quad lbl(e') = Previous$
where  $(n, n', n'')$  correspond to  $(tx, h_i, h_{i-1})$  respectively.