Towards Interoperability of Open and Permissionless Blockchains: A Cross-Chain Query Language

Felix Härer
Digitalization and Information Systems Group
University of Fribourg
Fribourg, Switzerland
E-Mail: felix.haerer@unifr.ch

***Abstract*—The rise of open and permissionless blockchains has introduced novel platforms for applications based on distributed data storage. At the application and business levels, long-established query languages such as SQL provide interoperability that can be complemented by blockchain-based data storage today, enabling permissionless and verifiable data storage along with decentralized execution across tens of thousands of nodes. However, when accessing one or more blockchains, interoperability is not provided today, posing challenges such as inhomogeneous data access in addition to different features and trade-offs, e.g. in data and distribution, scalability, and security. Towards interoperability in data access among the increasing number of blockchain platforms, this paper introduces a cross-chain query language for data access across blockchains. Similar to SQL, the language abstracts from implementation based on a data model compatible with the largest open and permissionless blockchains (OPB) today. The language, data model, and processing architecture are demonstrated and evaluated with an implemented prototype, aiming to contribute to the discussion on blockchain interoperability among OPB.**

***Index Terms*—Query Languages, Distributed Systems, Blockchains, Data Models, Interoperability**

I. Introduction

In August 2022, at least 22 well-known openly accessible blockchains exist with a significant number of active participants, processing their transactions over internet protocols[1]. These systems are, in their nature, open platforms for distributed data and applications, allowing applications of businesses or individuals to transact through a shared ledger without centralized coordination based on algorithmic consensus in the network [1]. Open systems with these components - blockchain data, network, consensus protocol - are verifiable in their operation and data by all participants, enabling novel decentralized applications such as programmable money or contracts [2]. In contrast to distributed systems of past decades, hundreds to thousands of nodes establish open and permissionless platforms. Estimations based on the connections of participating nodes count about 15000 nodes operating Bitcoin[2], 6000 operating Ethereum[3], and 3000 operating Cardano[4]; without accounting for additional nodes not visible due to their configuration or placement behind routers and firewalls. With the growing adoption, the increasing number of open and permissionless blockchains, and the vast amounts of openly available data, future relevance of these platforms for verifiable data storage and execution is assumed as a premise of this paper. Applications accessing the platforms include payments and currency, e-commerce, timestamping, and the attestation of data and links on the web [3]–[5].

Problem Statement. Software accessing data across open and permissionless blockchains (OPB) today faces challenges due to interoperability:

1) Inhomogeneous access to data due to various OPB implementations.
2) Different OPB data models and features exist.
3) Different OPB trade-offs exist, notably regarding scalability, security, and decentralization.

Research Objective and Contribution. The objective of this research is to address the three challenges towards higher interoperability between OPB. In the discussion on this topic, the paper contributes a cross-chain query language by specifying a common data model, a standardized syntax, and a processing architecture. For answering query statements written by users or applications, data is read from multiple blockchain nodes, integrated in the common data model, and processed according to the statements. Given the state-of-the-art of conceptual models and query languages suggested before, such as [6] and [7], the language design is derived abstract from implementation for several of the largest OPB today. The proof-of-concept implementation demonstrates feasibility and the potential for software to utilize OPB as part of their architecture.

Application Example. Several e-commerce websites might take part in commonly operated loyalty programs that issue reward points for customer purchases. Among other industries, these programs can be found for airlines working together[5].

This work is partially supported by the Swiss National Science Foundation project Domain-Specific Conceptual Modeling for Distributed Ledger Technologies [196889].

[1]https://messari.io/screener/active-adresses-15D9AE99

[2]https://bitnodes.io/

[3]https://ethernodes.org/

[4]https://adastat.net/pools/

[5]E.g. https://www.miles-and-more.com/ch/en.html

Given a cross-chain query language, business-level applications of different airlines could re-use and standardize data access, the underlying blockchain could be changed, and multiple blockchains of several loyalty programs could be used from individual applications.

The remainder of this paper is structured as follows. Section II introduces foundations and related work. Section III discusses OPB with their properties for the derivation of a data model. Subsequently, the data model, a grammar of the language syntax, and a processing architecture are derived. This approach is evaluated in Section IV with a prototype implementation utilizing multiple OPB. Section V concludes.

II. Foundations and Related Work

Blockchain foundations are introduced first, followed by open and permissioned blockchains and interoperability.

A. Blockchains

Following the initial publication of the Bitcoin whitepaper and software in 2008 and 2009, respectively [8], [9], the term *blockchain* has been introduced as a generalization of its technical architecture. The main components of (1.) a data structure of backward-linked blocks, (2.) a peer-to-peer network for data distribution, and (3.) a consensus protocol allow for novel properties. Notably, coordinating and validating all operations without centralized control, open access to all data and operations, and permissionless access where data and operations are not restricted to specific participants [2], [10]. Ethereum, and other blockchains following it, extended the capabilities with smart contracts as quasi Turing-complete programs [11], [12]. In addition to payments and money, smart contracts enable e-commerce, sales contracts, timestamping, and attestations, among other applications [3], [5], [13].

B. Open and Permissionless Blockchains

Growing development and adoption originating from Bitcoin and Ethereum have resulted in OPB with various properties. Table I lists five well-known OPB in order of their public number of network nodes, characterized by the properties of their data structure, network, and consensus protocols, as well as features related to smart contracts.

Data Structures. The initial design of backward-linked blocks in Bitcoin is combined in most other OPB with additional trees or graphs. In addition to transaction data from blocks, further queries must be carried out for non-transactional data or older data that has been pruned. For example, separate tree structures are present for state storage in Ethereum, where balances and smart contract variables can be retrieved [6].

Networks. Major OPB networks consist of approximately 1300 to 15000 nodes. Due to algorithmic operation and validation, higher node counts increase security, e.g. related to 51% attacks and selfish mining [14], [15] frequently observed in small Proof-of-Work systems such as *Bitcoin Gold* [15].

Consensus Protocols. In the protocols initially created between 2008 and 2022, a shift away from Proof-of-Work to Proof-of-Stake can be observed, introducing several trade-offs. While established blockchains such as Bitcoin and Ethereum have been emphasizing security and decentralization over the past years, efforts to improve efficiency and scalability are made in Cardano [16], Avalanche [17], and Solana [18]. The tendency is reflected in work on novel consensus protocols by the three blockchains, mostly based on Proof-of-Stake [19], [20] with higher efficiency and advantages to environmental impact. Avalanche and especially Solana emphasize scalability. In Solana, however, temporary protocol failures can be observed frequently, resulting in non-availability [21].

Smart Contract Features. For data queries and software applications, smart contract features are required. In this area, Bitcoin possesses a limited scripting language used for programmable monetary transactions and the scalable *lightning* overlay network [22]. The introduction of general-purpose programming in Ethereum and others introduces greater features and complexity. At present, most implementations are written and compiled for the Ethereum Virtual Machine, present in Ethereum and Avalanche. Cardano and Solana possess their own architectures with support for general-purpose programs.

C. Blockchain Interoperability

Interoperability is broadly recognized for transactions across blockchains in cross-chain swaps and similar concepts found in practice in so-called *bridges*. In addition, standardization efforts related to inhomogeneous data are beginning, not limited to query languages.

Cross-Chain Swaps. Swaps are commonly initiated through a protocol on an initial blockchain, where tokens or arbitrary data are locked to prevent further transfer in the beginning. A counterparty transaction is issued on a second blockchain to the initiator of the cross-chain swap, i.e., often another party pays for the tokens with another asset on the second chain. This transaction includes a cryptographic proof with a secret that unlocks the tokens on the initial chain. Finally, the counterparty withdraws tokens from the initial chain. Various protocols on this basis and variants exist [23], [24]. In atomic cross-chain swaps [25], [26], atomicity is provided for all transfers involved in a cross-chain swap. Implementations in bridges may, in practice, exhibit differing properties and assurances, not necessarily providing atomicity or other guarantees for the completion of the exchange. Bridges exist mainly for cryptocurrency exchange, e.g. Anyswap[6] and Connext[7] allow for cross-chain swaps between Ethereum, Avalanche, and others. Cross-chain swaps and bridges are not standardized and do not provide homogeneous access or queries.

Inhomogeneous Data. Standardization efforts address inhomogeneous data with few prior works related to inhomogeneous access. For Ethereum, [6] discusses a conceptual schema derived from the main data structures of the blockchain. [7] propose a query language for the content of blocks and transactions. The language design is based on SQL in its syntax

[6]https://anyswap.exchange/

[7]https://bridge.connext.network/

TABLE I
PROPERTIES OF WELL-KNOWN OPEN AND PERMISSIONLESS BLOCKCHAINS.

Blockchain	Data Structure	Network	Consensus Protocol	Smart Contract Features
[1] Bitcoin[a]	Blocks, UTXO data model	Bitcoin, approx. 15000 nodes	Nakamoto Consensus, Proof-of-Work	Stack-based script execution, monetary transactions
[2] Ethereum[b]	Blocks, account state storage in tree data structures	Ethereum Mainnet, approx. 6000 nodes	Ethash, memory-hard Proof-of-Work	Ethereum Virtual Machine, general-purpose programs
[3] Cardano[c]	Blocks, extended UTXO model	Cardano, approx. 3000 nodes	Ouroboros, Proof-of-Stake	General-purpose programs, functional
[4] Solana[d]	Block and graph data structures over different time spans	Solana Mainnet Beta, approx. 1600 nodes	Graph-based (proof--of-history), Proof-of-Stake	General-purpose programs
[5] Avalanche[e]	Block and graph data structures over different networks	Platform/Exchange/ Contract (P/X/C) chain, approx. 1300 nodes	Avalanche (P Chain) Snowman (X/C Chain), Proof-of-Stake	Ethereum Virtual Machine (C Chain), general-purpose programs

[a][18], https://bitnodes.io/
[b][29], https://ethereum.org/en/developers/docs/, https://ethernodes.org/
[c][15], https://adastat.net/pools/
[d][30], https://docs.solana.com, https://solanabeach.io/validators/
[e][23], https://stats.avax.network/dashboard/network-status/

and supports concepts such as projection and selection within Ethereum. For data analysis, a framework and implementation based on Scala has been proposed [27], where SQL or NoSQL is used with aggregation functions and similar analysis methods. The analysis approach [28] describes a data warehouse and ETL process for analyzing Ethereum data using standard SQL with a multi-dimensional data model for queries of dimension attributes and data aggregation support. This work and similar works might connect to multiple blockchains, however, they do not provide homogeneous data access, queries, or simultaneous access to data of multiple blockchains. Other works based on SQL include [29], using multiple blockchains for populating a standard MySQL database with the third-party service Google BigQuery. The use of third-party services as data sources presents another problem often observed in prior work, where validation of blockchain data is not possible or severely limited. Further approaches include public connectors between blockchains, blockchains that integrate with others, and hybrid approaches [30].

Limitations of Prior Work. In conclusion, present solutions are limited regarding (L1.) homogeneous data access, (L2.) standardized queries, (L3.) simultaneous access to multiple blockchains, or (L4.) blockchain data validation. Currently, the focus is on cross-chain swaps and siloed data analysis rather than data integration. The proposed query language addresses these limitations by suggesting a common data model (L1.), a standardized syntax (L2.), and a processing architecture (L3.) supporting local nodes (L4.) of multiple blockchains.

III. QUERY LANGUAGE

The two following subsections describe (A.) the data model and (B.) the syntax and processing architecture of the language. Query statements are processed according to the architecture in subsection (B.), resulting in instances of data model classes using data provided by APIs of local blockchain nodes.

A. Data Model

The language design is based on a data model integrating the main data structures and attributes of the OPB introduced in Section II-B. Based on prior work and existing tools discussed in Section II-C, classes and attributes of the five OPB have been identified, generalized, and integrated in a common data model. Figure 1 lays out the complete data model as UML class diagram. Table II lists the main model classes grouped into four packages for representing the chain, block, account, and transaction concepts of the OPB. For formulating queries, the syntax is introduced in subsection III-B. Statements are written in terms of the classes and attributes by specifying the source data with class and attribute names of the data model.

In the table and data model, the concepts of the OPB are represented by the following classes. The chain classes represent one main network and blockchain for Bitcoin, Ethereum, Cardano, and Solana, by the classes Chain, Network, and ChainDescriptor of the data model. Additional test networks with their separate blockchains, e.g. Ropsten and Görli in Ethereum, are represented by Network and ChainDescriptor. In Avalanche, the Network class encompasses one primary network, the first of potentially many *subnets*, with separate ChainDescriptor instances for the three P/X/C blockchains.

The Block and BlockDescriptor classes represent blocks with separate classes for the status of the block, the block validation through the consensus protocol, and the validators involved. Conceptually, blocks are described by an ID in the form of a hash value in all blockchains, with metadata such as timestamp and a height denoting the block number under the assumption no changes occur to non-final blocks. For example,

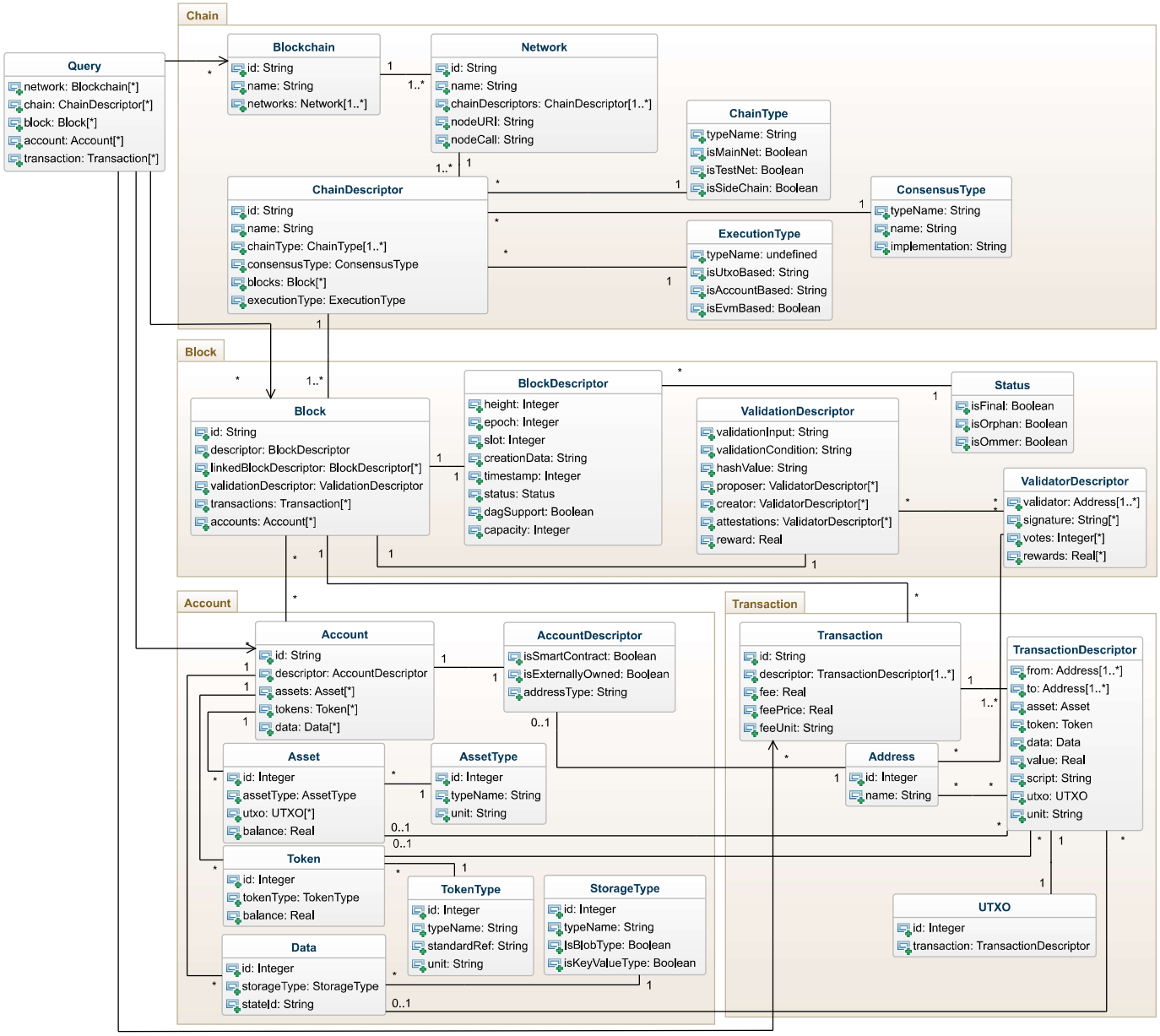

Fig. 1. Data model of the cross-chain query language as UML class diagram.

in Bitcoin, multiple blocks could be found as the successor to a given block; however, only one block will be included in the chain while the others will be discarded with *orphan* status. In contrast, the same case is handled in Ethereum by keeping one block in the main chain and keeping the other blocks at the same level with *ommer* status. Blocks are not explicitly finalized in Proof-of-Work chains, allowing for the declaration of *orphan* or *ommer* status for blocks found in parallel to prior blocks of the chain. However, the probability of existing blocks being replaced in this way decreases over time since multiple successive parallel blocks with greater cumulative work are required. An explicit finalization of blocks, preventing the occurrence of multiple successors, can be found in more recent Proof-of-Stake blockchains such as Solana.

Regarding data structure, blocks are linked to one or more existing blocks of the linkDescriptor attribute of the Block class, establishing either backward-linked blocks or a graph, such as a directed acyclic graph (DAG) in the Avalanche C chain. Blocks either contain transactions directly or are grouped into time-based slots and epochs for validation purposes with Proof-of-Stake. When appending a block, validation is carried out for each block or slot, requiring the participation of validators. In the ValidationDescriptor class, the creator of a block for Bitcoin and Ethereum provides validation for a linked block by the hashValue attribute. For the other Proof-of-Stake blockchains, proposers are recorded in the corresponding attributes with attestations, referencing the class ValidatorDescriptor. Each instance references any

TABLE II
DATA MODEL CLASSES SUPPORTING THE CONCEPTS OF THE OPB.

Chain classes	Block classes	Account classes	Transaction classes
Bitcoin chain	Blocks of transactions	-	Values (Bitcoin), UTXO
Ethereum chain	Blocks of transactions	Data storage, balances	Values (Ether), data incl. tokens
Cardano chain	Epochs, slots with blocks for transactions	Addresses, data	Values, assets, data, UTXO
Solana chain	Epochs, slots for transactions	Data storage	Data incl. tokens
Avalanche P/X/C chain	P/X: transaction DAG C: blocks of transactions	P/X: - C: Data storage, balances	X: values, assets, UTXO C: Data incl. tokens

number of appointed validators attesting the correctness of the block by means of their vote and signature. In this way, the concept of multiple groups of validators performing attestations is represented. When storing the transaction of a DAG, one or more transactions in a block can be linked to one or more transactions from a preceding block, indicated by the linkedBlockedDescriptor attribute in Block and the dagSupport attribute in BlockDescriptor that are set *true* in this case.

Accounts are a concept present in Ethereum, Solana, and Avalanche. Blocks contain accounts for the storage of assets, tokens, or data used for smart contracts. Notably, data might be used for the representation of assets or tokens directly, such as in Solana. Each account is described by an ID with the concept of an address being present in all blockchains. In an account, the storage of assets or tokens can refer to custom assets, such as in Cardano, or tokens represented by data in the general case. For tokens, token standards such as ERC-20 in Ethereum are represented by Token class attributes. Storing data uses binary large objects or key-value stores, utilized in hash-based mapping data structures.

Transaction concepts in Bitcoin and Cardano differ due to the lack of accounts in these blockchains. For this reason, transactions contain a reference to unspent transaction outputs (UTXOs) of previous transactions. In this model, a UTXO is included together with the transferred value and a script describing locking conditions or containing data. While the inclusion of data is implicit in Bitcoin, Cardano explicitly supports data in transactions and its storage linked to an address for smart contract functionality. In the case of Ethereum, Solana, and the Avalanche C chain, transactions are stored for the transfer of values, data, assets, or tokens between accounts. In the Avalanche X chain, the transfer of native assets is supported through the UTXO concept. In the data model, the attributes of Transaction and TransactionDescriptor allow for transfers between addresses by utilization of the attributes corresponding to the aforementioned concepts.

B. Language Syntax and Processing Architecture

The language syntax is based on established concepts of data query languages, in particular the Structured Query Language (SQL). On the one hand, the syntax of SQL and similar languages allow the representation of queries in a formalized way through relational algebra. On the other hand, queries and their articulation are accessible to domain experts without detailed knowledge of the underlying concepts. The SQL syntax is centered around the *SELECT-FROM-WHERE* block (SFW block). Based on English-language commands, the *SELECT* clause will perform a projection in the underlying relational model, semantically corresponding to columns, followed by the source of the relations in the *FROM* clause and the selection of tuples using conditions in the *WHERE* clause. In the relational model, set operations, and notably the Cartesian product, are the basis for all queries. For a cross-chain data language, these concepts are applied as follows.

Query Requirements. The syntax of query statements consists of query (Q), source (S), and filter (F) clauses:

- Q Query attributes are any attributes of the data model classes. Each attribute must be specified together with its class, determining one column of the query result for each source.
- S Sources specify blockchains and networks, optionally together with blocks, transactions, and accounts with assets, tokens, and data. In terms of the data model, each source must be specified by the attribute values of the identifying attributes of the Chain, Network, and ChainDescriptor classes. Optionally, an additional class, attribute, and attribute value of an identifying attribute from the classes Block, Transaction, Account, Asset, Token, or Data can be specified.
- F Filters are optional conditions filtering the query results by query attributes and sources. Each filter must be specified by a filter function for comparisons with two inputs using query attributes. On the result values from the query attributes, specified filters are applied in sequence.

Grammar. According to these requirements, Listing 1 partially shows the syntax definition[8]. In the grammar excerpt, the query statement syntax is described using the W3C variant of the Extended Backus Naur Form (EBNF) [31]. The query, source, and filter clauses specify projections, source data, and selections, respectively. In each clause, the specification of multiple values entails processing multiple result sets, in the case of SourceSpec for triggering the retrieval of data from multiple blockchains. Additionally, sources specify a chain, network, and chain descriptor with an optional block, transaction, or account according to the data model and requirements.

[8]Complete grammar: https://github.com/fhaer/CCQL/tree/main/grammar

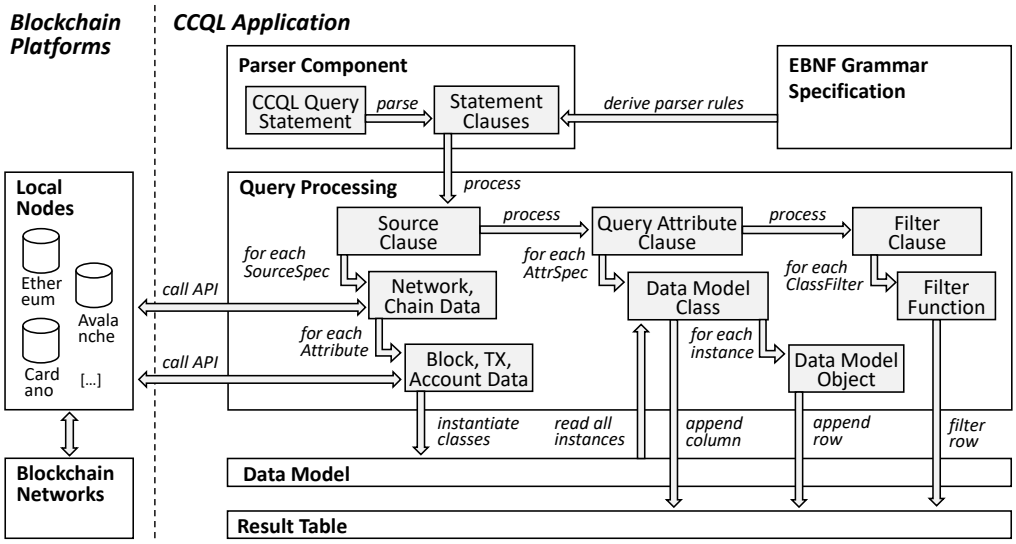

Fig. 2. Architecture for processing queries within an application.

Accounts storing assets, tokens, or data are accessed by the respective classes shown in the data model. The source and filter specifications are further detailed with the full EBNF grammar specification in the implementation.

```
QueryStatement ::=
  QueryAttrClause
  SourceClause
  FilterClause? ";"
QueryAttrClause ::=
  'Q ' AttrSpec ( ', ' AttrSpec )*
SourceClause ::=
  'S ' SourceSpec ( ', ' SourceSpec )*
FilterClause ::=
  'F ' FilterSpec ( ', ' FilterSpec )*
AttrSpec ::=
  CCQLClass '.' AttrName
SourceSpec ::=
  BlockchainI ':' NetI ':' ChainDescI
  (':' ( BlockI | TxI | AccI ) )?
FilterSpec ::=
  CCQLClass '.' AttrName ComparisonFunction IValue
```

Listing 1. EBNF excerpt. Attr: Attribute, Spec: Specification, Val: Value, Desc: Descriptor, I: Instance, Net: Network, Tx: Transaction, Acc: Account.

Processing Architecture. The processing of queries within the architecture is described in Figure 2. In an application connected to local nodes of blockchain platforms, query statements are issued to the parser component. After building the clauses, the query processing component first iterates the source clause, issuing for each SourceSpec and for each attribute API calls to the blockchain nodes. According to the classes and attributes, instances are created in the data model. Second, the query clause iterates each AttrSpec by selecting the corresponding classes from the model and producing columns in the result table. For the objects of the classes, rows are appended. Third, the filter clause applies each filter function. Finally, the result table contains the query result.

IV. Evaluation of Feasibility

The aim of this section is a demonstration of feasibility for the query language, its data model, and the processing architecture. For this purpose, an implementation compatible to the introduced OBP has been developed, consisting of a formal language grammar and a prototype application[9]. The grammar is realized with the Eclipse Modeling Framework and Xtext[10] to establish an external domain-specific language (DSL) based on it. In principle, the grammar is implementation independent and might be re-used in further applications. The language is implemented in a prototype command-line application together with the data model according to the proposed architecture, involving node access to the selected OPB for query execution. The prototype is written in Python 3.9 and utilizes the web3.py library for OPB access[11].

A. Software Setup

Setting up the application involved the following blockchain nodes with a configuration that fully validates all blocks:

- Bitcoin node: Bitcoin Core, version 22.0[12]. Initial data synchronization completed after 4 days.
- Ethereum node: go ethereum (geth[13]), version 1.10.510 with tracing and indexing of all transactions, and pruning of ancient block data. Initial data synchronization completed after approximately 12 weeks.
- Cardano node: Cardano node, version 1.34[14]. Initial data synchronization completed after approximately 2 days.
- Avalanche node: AvalancheGo, version 1.7[15]. Initial data synchronization completed after approximately 4 days.

The synchronizations were made on a PC with a AMD 3700X CPU, 32 GB RAM, and Samsung 980 Pro NVMe SSD, behind a 1 Gbit/s fiber internet connection. After the

[9]Available at https://github.com/fhaer/CCQL/tree/main
[10]https://www.eclipse.org/Xtext
[11]https://web3py.readthedocs.io/en/stable/
[12]https://bitcoin.org/de/download
[13]https://geth.ethereum.org/downloads/
[14]https://github.com/input-output-hk/cardano-node
[15]https://github.com/ava-labs/avalanchego/releases

```
$ ./ccql Q Block.id BlockDesc.height BlockDesc.timestamp Block.transactions S eth:main:1:Block.14505661 F BlockDesc.timestamp=1648887633
Create connection to Web3 Ethereum node ...
Parsing Web3 identity ...
Reading key file geth-data/keystore/UTC--2021-01-27T08-10-04.538339000Z--cbf332db0160fbe46c51f974577de404524f67c9

Query results:

1:Block.id|1:BlockDesc.height|1:BlockDesc.timestamp|1:Block.transactions|
0xfb2e9db3fd66722c5560f7b23ee38e1f8faf11ce4afd5b169b6c6f39bb678b4e|14505661|1648887633|0xc10a35453f2491dbcb1148182f83b73afe4c5b76267d68c7e8d8591e384f00ec|
0xfb2e9db3fd66722c5560f7b23ee38e1f8faf11ce4afd5b169b6c6f39bb678b4e|14505661|1648887633|0xf50fc4bece6589655cc366ff2638de0e2ed61e01b500c138d5d3cd5450327334|
[...]
0xfb2e9db3fd66722c5560f7b23ee38e1f8faf11ce4afd5b169b6c6f39bb678b4e|14505661|1648887633|0x09873bf8b8443ff248d43a7c910f45213fe428b86cb4fcbf90071658f8080cd8|
0xfb2e9db3fd66722c5560f7b23ee38e1f8faf11ce4afd5b169b6c6f39bb678b4e|14505661|1648887633|0x0abe49e5fcbb081d74c155aa14f1e44eaadb20fe19f3bf2021b9bdb16e41f6b9|

Number of rows: 17
```

Fig. 3. Query example 1.

```
$ ./ccql Q T.id TDesc.value TDesc.unit TDesc.data S eth:main:1:Transaction.0xf50fc4bece6589655cc366ff2638de0e2ed61e01b500c138d5d3cd5450327334 avax:main:c:
Transaction.0xb6b05f59ba6b69a082e6a8828d2afbf83a8847854fc56f3cc26a32e929104cef
Create connection to Web3 Ethereum node ...
Parsing Web3 identity ...
Reading key file geth-data/keystore/UTC--2021-01-27T08-10-04.538339000Z--cbf332db0160fbe46c51f974577de404524f67c9
Create connection to Avalanche node ...

Query results:

1:Transaction.id|1:TDesc.value|1:TDesc.unit|1:TDesc.data|2:Transaction.id|2:TDesc.value|2:TDesc.unit|2:TDesc.data|
0xf50fc4bece6589655cc366ff2638de0e2ed61e01b500c138d5d3cd5450327334|0.0|ETH|0xa9059cbb000000000000000000000071191c6199d5d5178822c5cc19fab83634929a5b00000
0000000000000000000000000d029cd740|0xb6b05f59ba6b69a082e6a8828d2afbf83a8847854fc56f3cc26a32e929104cef|0.0|AVAX|0xb88d4fde000000000000000000000000f15e18f
d289db61b8ab9f58ff64360086cc5061e000000000000000000000000f15e18fd289db61b8ab9f58ff64360086cc5061e0000000000000000000000000000000000000000000000000000000000000000
0036e5000000000000000080000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000|

Number of rows: 1
```

Fig. 4. Query example 2.

synchronizations, nodes and data were transferred to a typical end-user device, a laptop with AMD 5700U CPU, 16 GB RAM, and SK Hynix BC711 NVMe SSD. Both machines were running Ubuntu 21.10.

In order to demonstrate feasibility, query statements were evaluated with the prototype, as discussed in the following section. Each statement was executed on the laptop device, with local data retrieval from the blockchain node software according to the processing architecture. Given the local and fully-validating configuration of the node software, query results can be produced without network access. Query performance, therefore, does not depend on network latency and is limited only by local CPU and IO performance.

B. Prototype Operation

The architecture of the prototype is represented in Figure 2 and generally explained in Section III-B, *Processing Architecture*. In the following, two example queries are discussed.

The common task of locating transactions in a block is shown in Figure 3. The query attributes specify Block and BlockDescriptor (BlockDesc) classes with attributes of the block ID, Height, Timestamp, and transactions. Ethereum, its main network, chain 1, and block follow as source clause. Finally, a filter with a timestamp is applied. In the query results, data model classes and attributes such as Block.id and Block.height can be seen with their corresponding values, e.g. `0xfb2e[...]` and `14505661`, respectively.

With similar classes and attributes, the second example in Figure 4 shows a query common in a cross-chain swap scenario, retrieving transaction data from Ethereum and Avalanche. The query results show the attributes prefixed by the number of the source with instance-level data from the data model with corresponding values, with blocks and transaction identifiers as hexadecimal hash values.

C. Discussion

The prototype implements homogeneous data access to OPB, e.g. retrieving asset and data transfers from multiple blockchains. According to the grammar, data access is standardized and allows for statements involving one or more blockchains. For utilizing blockchain properties in a meaningful way, running blockchain nodes locally is required, involving substantial time and cost for initial synchronizations. The data model architecture follows a data integration approach, where data can be stored according to today's major OPB by populating relevant classes. In contrast, an approach of multiple individual data models alone would not address the problem at hand. General limitations of the prototype in its current stage are limited support for advanced concepts of the OPB, e.g. the calculation of transaction fees involving further utility tokens is outside the model. In terms of functionality, the queries with filters are limited in the prototype, only allowing for inner joins and equality comparisons.

V. Conclusion

In this paper, a cross-chain query language grammar, data model, and processing architecture are introduced for homogeneous data access across multiple blockchains. The approach supports homogeneous data access, standardization of queries, addressing of multiple blockchains in individual queries, and local validation of blockchain data. Prior research only covered these aspects partially. The feasibility of the approach could be positively evaluated with a prototype, despite functional limitations present in the implementation. In principle, software applications can utilize such an integrated approach for storing data on open and permissionless blockchains, independent of their implementation and openly available. Future research will further evaluate and extend the language design, data model, and architecture toward execution capabilities.

References

[1] M. Belotti, N. Bozic, G. Pujolle, and S. Secci, "A Vademecum on Blockchain Technologies: When, Which, and How," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3796–3838, 2019, 10.1109/COMST.2019.2928178.

[2] A. M. Antonopoulos and G. Wood, *Mastering Ethereum: Building Smart Contracts and DApps*. O'Reilly Media, 2019.

[3] A. Narayanan and J. Clark, "Bitcoin's academic pedigree," *Communications of the ACM*, vol. 60, no. 12, pp. 36–45, 2017, 10.1145/3132259.

[4] I. Weber and M. Staples, "Programmable money: Next-generation conditional payments using blockchain," in *11th International Conference on Cloud Computing and Services Science - CLOSER,*, INSTICC. SciTePress, 2021, pp. 7–14, 10.5220/0010535800070014.

[5] F. Härer and H.-G. Fill, "Decentralized attestation and distribution of information using blockchains and multi-protocol storage," *IEEE Access*, vol. 10, pp. 18 035–18 054, 2022, 10.1109/ACCESS.2022.3150356.

[6] A. Olivé, "The Conceptual Schema of Ethereum," in *Conceptual Modeling*, G. Dobbie, U. Frank, G. Kappel, S. W. Liddle, and H. C. Mayr, Eds. Cham: Springer International Publishing, 2020, pp. 418–428, 10.1007/978-3-030-62522-1_31.

[7] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, "Ethereum query language," in *1st International Workshop on Emerging Trends in Software Engineering for Blockchain*. ACM, 2018, 10.1145/3194113.3194114.

[8] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," Tech. Rep., 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[9] ——, "Bitcoin - A software-based online payment system," 2009. [Online]. Available: https://sourceforge.net/p/bitcoin/news/

[10] J. Garay, A. Kiayias, and N. Leonardos, "The Bitcoin Backbone Protocol: Analysis and Applications," in *EUROCRYPT 2015*. Springer, 2015, 10.1007/978-3-662-46803-6.

[11] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum, Tech. Rep., 2022. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[12] V. Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," Tech. Rep., 2014. [Online]. Available: https://ethereum.org/en/whitepaper/

[13] J. Ladleif and M. Weske, "A Unifying Model of Legal Smart Contracts," in *Conceptual Modeling*. Cham: Springer International Publishing, 2019, pp. 323–337, 10.1007/978-3-030-33223-5_27.

[14] M. K. Shrivas, T. Y. Dean, and S. S. Brunda, "The Disruptive Blockchain Security Threats and Threat Categorization," in *2020 First International Conference on Power, Control and Computing Technologies (ICPC2T)*. Raipur, India: IEEE, 2020, 10.1109/ICPC2T48082.2020.9071475.

[15] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, and D. Mohaisen, "Exploring the Attack Surface of Blockchain: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, 2020, 10.1109/COMST.2020.2975999.

[16] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol," Tech. Rep. 889, 2016. [Online]. Available: https://eprint.iacr.org/2016/889

[17] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, "Scalable and Probabilistic Leaderless BFT Consensus through Metastability," *arXiv:1906.08936 [cs]*, 2020, 10.48550/arXiv.1906.08936.

[18] A. Yakovenko, "Solana: A new architecture for a high performance blockchain v0.8.14," Solana, Tech. Rep., 2018. [Online]. Available: https://github.com/solana-labs/whitepaper/blob/master/solana-whitepaper-en.pdf

[19] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine Agreements for Cryptocurrencies," in *26th Symposium on Operating Systems Principles*, ser. SOSP '17. ACM, 2017, 10.1145/3132747.3132757.

[20] Ethereum, "Proof-of-stake (PoS)," 2022. [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/

[21] A. Hayward, "Solana Blames 'Denial of Service Attack' for Last Week's Downtime," 2021. [Online]. Available: https://decrypt.co/81375/solana-blames-denial-of-service-attack-for-last-weeks-downtime

[22] A. M. Antonopoulos, R. Pickhardt, and O. Osuntokun, *Mastering the Lightning Network*. O'Reilly Media, 2021.

[23] B. Pillai, K. Biswas, and V. Muthukkumarasamy, "Cross-chain interoperability among blockchain-based systems using transactions," *The Knowledge Engineering Review*, vol. 35, 2020, 10.1017/S0269888920000314.

[24] N. Shadab, F. Houshmand, and M. Lesani, "Cross-chain Transactions," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2020, 10.1109/ICBC48266.2020.9169477.

[25] M. Herlihy, "Atomic Cross-Chain Swaps," in *2018 ACM Symposium on Principles of Distributed Computing*. Egham United Kingdom: ACM, 2018, pp. 245–254, 10.1145/3212734.3212736.

[26] V. Zakhary, D. Agrawal, and A. El Abbadi, "Atomic commitment across blockchains," *Proceedings of the VLDB Endowment*, vol. 13, no. 9, pp. 1319–1331, 2020, 10.14778/3397230.3397231.

[27] M. Bartoletti, S. Lande, L. Pompianu, and A. Bracciali, "A general framework for blockchain analytics," in *1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, ser. SERIAL '17. New York, NY, USA: ACM, 2017, pp. 1–6, 10.1145/3152824.3152831.

[28] G. Camozzi, F. Härer, and H.-G. Fill, "Multidimensional Analysis of Blockchain Data Using an ETL-based Approach," in *Wirtschaftsinformatik 2022 Proceedings*, 2022.

[29] I. Liiv, "Exploration with Structured Query Language," in *Data Science Techniques for Cryptocurrency Blockchains*, ser. Behaviormetrics: Quantitative Approaches to Human Behavior. Springer, 2021. [Online]. Available: http://doi.org/10.1007/978-981-16-2418-6_2

[30] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A Survey on Blockchain Interoperability: Past, Present, and Future Trends," *arXiv:2005.14282 [cs]*, 2021, 10.48550/arXiv.2005.14282.

[31] W3C, "Blindfold Grammars," 2001. [Online]. Available: https://www.w3.org/2001/06/blindfold/grammar