POnto ontology evolution

Query analysis and extension

Technical Report

Draft version 1.0 – September 2nd, 2023

Web3 Foundation Grants Program

Contributors

Rafael Brandão (rafael@mobr.ai)
Marcio Moreno (mmoreno@mobr.ai)

MOBR SYSTEMS

1. Introduction

POnto is a Polkadot ontology designed to represent and relate the main entities of the Polkadot ecosystem. Its focus is to support developers, researchers, and enthusiasts to enhance data analysis, communicability and domain knowledge sharing within the Polkadot community.

Aiming at evolving with the current version of POnto, we targeted the support for various aspects related to the queries available in this RFP (Data Analysis Tools for Substrate-based Blockchains), as well as query examples from the Substrate-ETL project repo and this medium post about it. Table 1 lists the queries that guided the proposed POnto extension.

Table 1. List of queries to be supported with the proposed POnto extension.

#	Query	From
1	Which transactions / accounts were responsible for the reserved balance in an account?	RFP (Data Analysis Tools for Substrate-based Blockchains)
2	What modules currently depend on consumers, providers, and sufficient reference counters for a certain account, and which transactions introduced/removed those references?	RFP (Data Analysis Tools for Substrate-based Blockchains)
3	Which accounts have delegated OpenGov votes to an account or to which accounts the account in question has delegated their votes to for each track, taking into account indirect delegations too (e.g. Account A delegates to Account B which delegates to Account C)?	RFP (Data Analysis Tools for Substrate-based Blockchains)
4	How many distinct XCM Transfer senders and beneficiaries were there in 2022 in Polkadot vs Kusama?	Substrate-ETL medium post

5	How many distinct assets were transferred in Polkadot and Kusama and how much was transferred?	Substrate-ETL medium post
6	How many distinct parachains are sending XCM Transfers in Polkadot vs Kusama?	Substrate-ETL medium post
7	How many distinct pallet sections/methods were used for sending XCM Transfers in Polkadot vs Kusama?	Substrate-ETL medium post
8	How many XCM Transfers happened in 2022?	Substrate-ETL medium post
9	(modified) How many distinct extrinsicIDs XCM Transfers successfully completed in 2022 across Polkadot and Kusama?	Substrate-ETL medium post
10	Get blocks of paraid 2000	Substrate-ETL github
11	Get extrinsics of paraid 2000	Substrate-ETL github
12	Get XCM Transfers of Polkadot Network	Substrate-ETL github

2. Query analysis

This section delves into a systematic examination of the queries originating from the Web3 Foundation Grants Program's Request for Proposals (RFP) and the Substrate-ETL project. This analysis aims to evaluate the compatibility of these queries with the existing POnto ontology and identify potential areas for ontology expansion. By scrutinizing concepts, relationships, and linguistic elements within the queries, this section provides insights into how the POnto ontology can effectively accommodate information commonly desired from the Polkadot ecosystem.

2.1 Qualitative coding analysis

To assess and identify concepts and relationships on the set of examined queries, we conducted a qualitative coding analysis. This analysis is a systematic process used in qualitative research to categorize and interpret data by identifying patterns, themes, and meaningful segments known as codes. This process allows for an understanding of human rationale and complex phenomena by extracting meaningful insights from the data. The legend below illustrates the "codes" considered during the query analysis.

Legend:

☐ Concept already in POnto ☐ Missing concept ☐ Relationship, connectors, qualifiers ☐ Commands

☐ Instances / individuals

From the RFP:

1. Which transactions/accounts were responsible for the reserved balance in an account?

2. What modules currently depend on consumers, providers, and sufficients reference counters for a certain account, and which transactions introduced/removed those references?

3. Which accounts have delegated OpenGov votes to an account or to which accounts the account in question has delegated their votes to for each track, taking into account indirect delegations too (e.g. Account A delegates to Account B which delegates to Account C)?

Accounts have some provenance information that is pretty difficult or currently impossible to extract in block explorers. The account reference counter, account balance reserved provenance (see: https://docs.substrate.io/reference/account-data-structures/).

From Substrate-ETL medium post:

1. How many distinct XCM Transfer senders and beneficiaries were there in 2022 in Polkadot vs Kusama?

2. How many distinct assets were transferred in Polkadot and Kusama and how much was transferred?

3. How many distinct parachains are sending XCM Transfers in Polkadot vs Kusama?

4. How many distinct pallet sections/methods were used for sending XCM Transfers in Polkadot vs Kusama?

5. How many XCM Transfers happened in 2022?

6. (modified). How many distinct extrinsicIDs XCM Transfers successfully completed in 2022 across Polkadot and Kusama?

From Substrate-ETL github:

1. Get blocks of paraid 2000

2. Get extrinsics of paraid 2000

3. Get XCM Transfers of Polkadot Network

2.2 Summarization

This section summarizes identified entities and relevant terms of the analyzed queries. Table 1 lists missing concepts, Table 2 lists relationship terms, Table 3 lists connectors and qualifier terms, and Table 4 shows a list of command expressions observed in the queries.

Table 1. List of missing concepts identified in analyzed queries.

Concept	Comment	Extension in POnto
Account reference counters	The account reference counters[1] track account dependencies in the runtime. For example, if you store data under a map controlled by an account, you wouldn't want to delete the account until the data stored under the map the account controls has been deleted.	One new owl:Class ponto:ReferenceCounter With three new rdfs:subClass ponto:Sufficients ponto:Consumers ponto:Providers

[1] https://docs.substrate.io/reference/account-data-structures/#account-reference-counters

	Runtime developers can update these counters using methods exposed by the frame-system pallet.	
Asset	Related to ponto:typeOfAsset	Add new owl:Class: ponto:Asset Set token as a subclass of ponto:Asset
Beneficiary	Indicates the recipient of a transfer or referendum value	New owl:Class ponto:Beneficiary
Consumer reference counter	The number of other modules that currently depend on this account's existence. The account cannot be reaped until this is zero.	One new owl:Class ponto:ReferenceCounter With three new rdfs:subClass ponto:Consumers
Pallet sections/methods	Related to ponto:Pallet	New owl:Class as rdfs:subClass of ponto:Pallet ponto:PalletSection New owl:Class as rdfs:subClass of ponto:PalletSection ponto:PalletMethod
paraid (paraId, para_id, parachainId?)	Related to ponto:Parachain Related to ponto:hasParaId	POnto already has a literal for this. We will map related terms in the CNL. It will be possible to refer to Parachains by their paraIds.
Provider reference counter	The number of other modules that allow this account to exist. The account may not be reaped until this (providers) and `sufficients` are both zero.	One new owl:Class ponto:ReferenceCounter With three new rdfs:subClass ponto:Providers
Reserved balance	Tokens can be reserved for various reasons that use space in the chain state. Unlike locks, reserves do stack. So, each new action that requires a deposit reserves the necessary amount added to any previous reserves that might exist. Additionally, reserved tokens can't be used for other purposes. Some examples of common reserves are in order to create an on-chain identity or sub-identity, set up a proxy account, or initiate a multisig transaction. In these examples, if the on-chain identity or the proxy is removed, or when the multisig transaction is completed or canceled, the reserve is released and becomes transferable balance	Three new owl:DatatypeProperty ponto:reserved ponto:locked ponto:transferrable

	again. Related to ponto:Token, ponto:hasBalance	
Sufficients reference counter	The sufficients reference counter indicates if an account is self-sufficient and can exist by itself. For example, in the Assets pallet, an account can have sufficient number of certain assets but without owning any native account balance.	One new owl:Class ponto:ReferenceCounter With three new rdfs:subClass ponto:Sufficients
XCMTransfer, extrinsicIDs	Related to ponto:XCM Related to ponto:Transaction Related to ponto:Extrinsic	New rdf:Property ponto:amountXCMTransferred It will be possible to refer to Transactions by their unique IDs as well.

Table 2. List of relationship terms identified in analyzed queries.

Relationship terms	Comment	Extension in POnto		
		owl:ObjectProperty	domain	range
delegatesTo, hasDelegated, haveDeletaged	Related to ponto:VoteDelegation	ponto:delegatesVote Inverse: votesDelegatedBy	ponto:Account	ponto:Account
dependsOn	Related to ponto:Pallet	ponto:dependsOn	ponto:Runtime Module	One new owl:Class ponto:Reference Counter With three new rdfs:subClass ponto:Sufficients ponto:Consumer s ponto:Providers
introduces, removes	Related to reference counters, using the inc_consumers(), dec_consumers(), inc_providers(), dec_providers(), inc_sufficients(), and dec_sufficients() methods exposed by the frame-system pallet	ponto:incrementsRefC ount ponto:decrementsRef Count	ponto:Transacti on	new owl:Class ponto:Reference Counter
responsibleFor	Transactions/accounts	ponto:hasTranferableB	ponto:Token	One new

	responsible for reserved balances	alance ponto:hasReservedBalance ponto:hasLockedBalance		owl:Class Balance Three new rdfs:subClassOf ponto:Balance ponto:ReservedBalance ponto:LockedBalance ponto:TransferableBalance
hasBeneficiary	Indicates the beneficiary of a transfer or referendum value	new owl:ObjectProperty ponto:hasTransferBeneficiary	ponto:Transfer	ponto:Beneficiary
		new owl:ObjectProperty ponto:hasReferendumBeneficiary	ponto:Referendum	ponto:Beneficiary
hasLockType	Locked tokens are usually accompanied by an unlocking period, a "cooldown" period before the lock expires after the reason for the lock has ceased to exist.	ponto:hasLockType	ponto:LockedBalance	One new owl:Class ponto:LockType With four new rdfs:subClassOf ponto:LockReason ponto:StakingLockType ponto:DemocracyLockType ponto:ElectionLockType ponto:VestingLockType
reservedFor	Tokens can be reserved for various reasons that use space in the chain state. Unlike locks, reserves do stack. So, each new action that requires a deposit reserves the necessary amount added to any previous reserves that might exist.	New owl:DatatypeProperty ponto:reservedFor	ponto:ReservedBalance	New owl:Class ponto:ReservationReason
sending, areSending, isSending	Related to ponto:Sender, ponto:hasSender	ponto:amountXCMTransferred Synonyms and	ponto:XCMTransaction	xsd:decimal

		equivalent terms can be addressed using NLP mechanisms		
usedFor	Relates pallets to transactions	ponto:hasOriginationSection ponto:hasOriginationMethod	ponto:Transaction	ponto:PalletSection ponto:PalletMethod
votesTo	Related to ponto:VotingPower, ponto:voting	New owl:Class ponto:Vote	-	-
		New owl:ObjectProperty ponto:hasVoter	ponto:Vote	ponto:Account
		New owl:DatatypeProperty ponto:hasAmount	ponto:Vote	Literal xsd:decimal
		New owl:DatatypeProperty ponto:hasConviction	ponto:Vote	Literal xsd:decimal
		New owl:DatatypeProperty ponto:hasDecision	ponto:Vote	New owl:DatatypeProperties ponto:aye ponto:nay ponto:abstained
wasTransfered, wereTransfered	Related to ponto:Transaction	Synonyms and equivalent terms can be addressed using NLP mechanisms		

Table 3. List of connector and qualifier terms identified in analyzed queries.

Connectors and qualifiers	Comment
across	Relates two or more entities
and, considering, takingIntoAccount	Concatenates multiple sub-queries in a complex query Synonyms and equivalent terms can be addressed using NLP mechanisms
completedIn	Considers there is a timestamp for the referred entities
distinct	Qualifier keyword to eliminate duplicates
for, forEach	Relates one or more classes and individuals, time interval

in (on, at)	Refers to classes, instances, time instant or interval
of (from)	May relate classes and individuals as a whole, or specific attributes of them (c.f. Substrate-ETL github queries). Can also be used to refer to time intervals.
successfullyCompleted	Considers there is a state for the referred entities
vs	Comparison between two entities

Table 4. List of command expressions identified in analyzed queries.

Command	Comment
get	Get one or more entities
how many	Count entities
how much	Retrieve or sum up numerical attributes
what	Get one or more entities Define an entity
which	Get one or more entities

3. SPARQL query translation

Going forward with the proposed extensions to the POnto ontology, this section presents SPARQL queries equivalents for the natural language queries analyzed in the previous section of this report. The translated queries serve as conceptual validators and support the verification over the completeness of the proposed extension.

Queries From the RFP:

```
# Which transactions/accounts were responsible for the reserved balance in an account?

SELECT DISTINCT ?resp {
  ?param_account a ponto:Account ;
     ponto:hasAddress "14UgDLpYd8E28QEmnaT4ZJTsM2kcm84Zo2EoD7wDv8S5CE4w"^^xsd:string ;
     ponto:hasReservedBalance ?rb .
  {
     ?account a ponto:Account ;
        ponto:hasReserved ?rb .
     BIND (?account as ?resp)
  }
  UNION
  {
     ?transaction a ponto:Transaction ;
        ponto:hasReserved ?rb .
     BIND (?transaction as ?resp)
  }
}
```

Listing 1. SPARQL query equivalent for "Which transactions/accounts were responsible for the reserved balance in an account?".

```
# What modules currently depend on consumers, providers, and sufficients reference counters for a certain
account, and which transactions introduced/removed those references?

SELECT DISTINCT ?module ?transaction WHERE {
  ?param_account a ponto:Account ;
    ponto:hasAddress "14UgDLpYd8E28QEmnaT4ZJTsM2kcm84Zo2EoD7wDv8S5CE4w"^^xsd:string ;
    ponto:hasConsumers ?consumers ;
    ponto:hasProviders ?providers ;
    ponto:hasSufficients ?suficients .

  { ?module ponto:dependsOn ?consumers . } UNION { ?module ponto:dependsOn ?providers . } UNION {
?module ponto:dependsOn ?sufficients . }
  { ?transaction ponto:incrementsRefCount ?consumers . } UNION { ?transaction ponto:incrementsRefCount
?providers . } UNION { ?transaction ponto:incrementsRefCount ?sufficients . } UNION
  { ?transaction ponto:decrementsRefCount ?consumers . } UNION { ?transaction ponto:decrementsRefCount
?providers . } UNION { ?transaction ponto:decrementsRefCount ?sufficients . }
}
```

Listing 2. SPARQL query equivalent for "What modules currently depend on consumers, providers, and sufficients reference counters for a certain account, and which transactions introduced/removed those references?".

```
# Which accounts have delegated OpenGov votes to an account or to which accounts the account in question
has delegated their votes to for each track, taking into account indirect delegations too (e.g. Account A
delegates to Account B which delegates to Account C)?

SELECT ?track ?delegatorAccount ?voterAccount
WHERE {
 ?vote ponto:hasVoterAccount ?voterAccount .
 ?vote ponto:voteDelegatedBy ?delegatorAccount .
 ?referendum ponto:hasVote ?vote .
 ?referendum ponto:hasReferendumTrack ?track .
}
```

Listing 3. SPARQL query equivalent for "Which accounts have delegated OpenGov votes to an account or to which accounts the account in question has delegated their votes to for each track, taking into account indirect delegations too (e.g. Account A delegates to Account B which delegates to Account C)?".

Queries from Substrate-ETL medium post:

```
# How many distinct XCM Transfer senders and beneficiaries were there in 2022 in Polkadot vs Kusama?

SELECT ?relaychain (COUNT(DISTINCT ?sender) AS ?senders) (COUNT(DISTINCT ?beneficiary) AS
?beneficiaries) {
	?transfer ponto:recordedOn ?block .
	{ ?block ponto:composes ponto:Polkadot . BIND (ponto:Polkadot as ?relaychain) }
	UNION
	{ ?block ponto:composes ponto:Kusama . BIND (ponto:Kusama as ?relaychain) }
	?transfer a ponto:XCMTransfer .
	?transfer ponto:hasOriginationTimestamp ?timestamp .
	?transfer ponto:hasSender ?sender .
	?transfer ponto:hasTransferBeneficiary ?beneficiary .
	?transfer ponto:hasDestinationExecutionStatus "success"^^xsd:string .
	FILTER(?timestamp > "2022-01-01T00:00:00Z"^^xsd:dateTimeStamp && ?timestamp <
"2022-12-31T23:59:59Z"^^xsd:dateTimeStamp)
}
GROUP BY ?relaychain
```

Listing 4. SPARQL query equivalent for "How many distinct XCM Transfer senders and beneficiaries were there in 2022 in Polkadot vs Kusama?".

```
# How many distinct assets were transferred in Polkadot and Kusama and how much was transferred?

SELECT ?symbol
	(ROUND(SUM(?origination_amount_sent)) AS ?total_origination_amount_sent)
	(ROUND(SUM(?destination_amount_received)) AS ?total_destination_amount_received)
	(ROUND(SUM(?origination_amount_sent_usd)) AS ?total_origination_amount_sent_usd)
	(ROUND(SUM(?destination_amount_received_usd)) AS ?total_destination_amount_received_usd)
	(ROUND(AVG(?origination_amount_sent_usd)) AS ?avg_origination_amount_sent_usd)
	(ROUND(AVG(?destination_amount_received_usd)) AS ?avg_destination_amount_received_usd)
	(COUNT(*) AS ?num_records)
WHERE {
    ?transfer a ponto:XCMTransfer ;
            ponto:hasOriginationTimestamp ?origination_ts ;
            ponto:hasDestinationExecutionStatus "success" ;
            ponto:hasTransferAsset ?asset ;
            ponto:hasTransferAmountSent ?origination_amount_sent ;
            ponto:hasTransferAmountReceived ?destination_amount_received ;
            ponto:hasTransferAmountSentUSD ?origination_amount_sent_usd ;
            ponto:hasTransferAmountReceivedUSD ?destination_amount_received_usd .
    ?asset ponto:hasSymbol ?symbol .

    FILTER(?origination_ts >= "2022-01-01"^^xsd:dateTime && ?origination_ts <= "2022-12-31"^^xsd:dateTime)
}

GROUP BY ?symbol
ORDER BY DESC(?total_origination_amount_sent_usd) DESC(?num_records)
```

Listing 5. SPARQL query equivalent for "How many distinct assets were transferred in Polkadot and Kusama and how much was transferred?".

```
# How many distinct parachains are sending XCM Transfers in Polkadot vs Kusama?

SELECT ?relaychain (COUNT(DISTINCT ?parachain) AS ?parachains) {
        ?transfer ponto:recordedOn ?block .
        { ?block ponto:composes ponto:Polkadot . BIND (ponto:Polkadot as ?relaychain) }
        UNION
        { ?block ponto:composes ponto:Kusama . BIND (ponto:Kusama as ?relaychain) }
        ?transfer a ponto:XCMTransfer .
        ?transfer ponto:hasOriginationTimestamp ?timestamp .
        ?transfer ponto:hasSender ?sender .
                        ?sender ponto:recordedOn ?parablock .
                        ?parablock ponto:composes ?parachain .
        ?transfer ponto:hasDestinationExecutionStatus "success"^^xsd:string .
        FILTER(?timestamp > "2022-01-01T00:00:00Z"^^xsd:dateTimeStamp && ?timestamp <
"2022-12-31T23:59:59Z"^^xsd:dateTimeStamp)
}
GROUP BY ?relaychain
```

Listing 6. SPARQL query equivalent for "How many distinct parachains are sending XCM Transfers in Polkadot vs Kusama?".

```
# How many distinct pallet sections/methods were used for sending XCM Transfers in Polkadot vs Kusama?

SELECT ?relaychain (COUNT(DISTINCT ?section) AS ?sections) (COUNT(DISTINCT ?method) AS
?methods) {
        ?transfer ponto:recordedOn ?block .
        { ?block ponto:composes ponto:Polkadot . BIND (ponto:Polkadot as ?relaychain) }
        UNION
        { ?block ponto:composes ponto:Kusama . BIND (ponto:Kusama as ?relaychain) }
        ?transfer a ponto:XCMTransfer .
        ?transfer ponto:hasOriginationSection ?section .
        ?transfer ponto:hasOriginationMethod ?method .
        ?transfer ponto:hasOriginationTimestamp ?timestamp .
        ?transfer ponto:hasDestinationExecutionStatus "success"^^xsd:string .
}
GROUP BY ?relaychain
```

Listing 7. SPARQL query equivalent for "How many distinct pallet sections/methods were used for sending XCM Transfers in Polkadot vs Kusama?".

```
# How many XCM Transfers happened in 2022?

SELECT (COUNT(?transfer) AS ?transfers) {
        ?transfer a ponto:XCMTransfer .
        ?transfer ponto:hasOriginationTimestamp ?timestamp .
        ?transfer ponto:hasDestinationExecutionStatus "success"^^xsd:string .
        FILTER(?timestamp > "2022-01-01T00:00:00Z"^^xsd:dateTimeStamp && ?timestamp <
"2022-12-31T23:59:59Z"^^xsd:dateTimeStamp)
}
```

Listing 8. SPARQL query equivalent for "How many XCM Transfers happened in 2022?".

```
# How many distinct extrinsicIDs XCM Transfers successfully completed in 2022 across Polkadot and Kusama?

SELECT ?relaychain (COUNT(DISTINCT ?extrinsicId) AS ?ids) {
        ?extrinsic ponto:hasExtrinsicId ?extrinsicId .
        ?extrinsic ponto:hasXCMTransfer ?transfer .
        ?transfer ponto:hasOriginationTimestamp ?timestamp .
        ?transfer ponto:hasDestinationExecutionStatus "success"^^xsd:string .
        ?transfer ponto:recordedOn ?block .
        { ?block ponto:composes ponto:Polkadot . BIND (ponto:Polkadot as ?relaychain) }
        UNION
        { ?block ponto:composes ponto:Kusama . BIND (ponto:Kusama as ?relaychain) }
        FILTER(?timestamp > "2022-01-01T00:00:00Z"^^xsd:dateTimeStamp && ?timestamp <
"2022-12-31T23:59:59Z"^^xsd:dateTimeStamp)
}
GROUP BY ?relaychain
```

Listing 9. SPARQL query equivalent for "How many distinct extrinsicIDs XCM Transfers successfully completed in 2022 across Polkadot and Kusama?".

Queries from Substrate-ETL github:

```
# Get blocks of paraid 2000

SELECT ?blocks WHERE {
    ?blocks ponto:composes ?parachain .
    ?parachain ponto:hasParaId 2000 .
}
```

Listing 10. SPARQL query equivalent for "Get blocks of paraid 2000"

```
# Get extrinsics of paraid 2000

SELECT ?extrinsic WHERE {
    ?parachain ponto:hasParaId 2000 .
    ?block ponto:composes ?parachain ;
        ponto:hasExtrinsic ?extrinsic .
}
```

Listing 11. SPARQL query equivalent for "Get extrinsics of paraid 2000"

```
# Get XCM Transfers of Polkadot Network

SELECT ?xcmtransfer WHERE {
    ?block ponto:composes ponto:Polkadot ;
        ponto:hasExtrinsic ?extrinsic .
    ?extrinsic ponto:hasXCMTransfer ?xcmtransfer .
}
```

Listing 12. SPARQL query equivalent for "Get XCM Transfers of Polkadot Network"

4. Final remarks

This report presents an analysis and extension of the POnto ontology considering the alignment with queries from a Web3 Foundation Grants Program's RFP as well as the Substrate-ETL project.

POnto's goal is to represent Polkadot ecosystem entities for effective data analysis and knowledge sharing. Through a coding analysis, we identified missing concepts and relevant relationships, leading to extensions to the ontology, introducing new classes, properties, and relationships to bridge the point out gaps.

This work contributes to the ontology's evolution, aiming at supporting enhanced data analysis and communication within the Polkadot community. It is a step towards strengthening the POnto ontology's utility in addressing specific query needs for the envisioned Polkadot Analytics Platform (PAP).

Future refinements and extensions to the proposed ontology are expected. We encourage engagement and participation, suggestions from the community are key for POnto's evolution.

References

Polkadot Analytics Platform (Stage 1) Grant: https://github.com/w3f/Grants-Program/pull/1883

RFP:
https://github.com/w3f/Grants-Program/blob/master/docs/RFPs/Under%20Development/data_analysis_tools.md

Substrate-ETL: https://github.com/colorfulnotion/substrate-etl

Accounts balances and locks:
https://support.polkadot.network/support/solutions/articles/65000182717-account-balances-and-locks

Substrate Docs – Account data structures:
https://docs.substrate.io/reference/account-data-structures/

Acknowledgement

This work was supported by a grants program from the Web3 Foundation and is publicly available.