Protocol Conformance of Collaborative SPARQL using Multiparty Session Types

Ari Hernawan$^{[0009-0001-2465-424X]}$ and Nobuko Yoshida$^{[0000-0002-3925-8557]}$

University of Oxford, Oxford, UK
`{ari.hernawan,nobuko.yoshida}@cs.ox.ac.uk`

Abstract. Decentralised linked data gives users rights over their data while being accessible to other domains. The RDF (Resource Description Framework) and SPARQL have been the standard specifications for managing linked data for several years. Recent research and development introduce scalable, centralised and distributed RDF store engines with the SPARQL. However, writing SPARQL federated queries may grow more complex as the number of domain participants increases, presenting challenges such as source discovery, completeness and performance. This paper presents a SPARQL Query Template (SQT) that applies Multiparty Session Types (MPST) to determine the order of federated queries. We also guarantee protocol conformance between MPST and SPARQL relational algebra.

Keywords: Multiparty session types · SPARQL · Federated Query · Protocol Conformance

1 Introduction

The RDF (Resource Description Framework) and SPARQL 1.1[1] are the two W3C recommended standards for querying and manipulating linked data on the Web. Since SPARQL's introduction in 2008, significant research and technological advancements have supported RDF and SPARQL specifications. Many RDF repositories, such as DBpedia, UniProt (Universal Protein Resource) and Rhea, are publicly available to provide information from diverse domains [1–6]. DBpedia, for example, contains around 900 million triplets as of January 2021[2] and maintains a distributed infrastructure that extracts approximately 5500 triplets per second and 21 billion triples per release. In real-world applications, RDF datasets are often partitioned and built upon distributed environments [7].

There has been rigorous work on handling massive RDF datasets. One way to manage the datasets is by partitioning and distributing them under the same RDF engine or by decentralising them on domain-specific repositories. The latest distributed RDF engine implemented with a custom communication protocol has shown superior performance compared to other centralised or MapReduce-based engines [7–16]. In addition, SPARQL 1.1 federated query allows merging

[1] `https://www.w3.org/TR/sparql11-overview/`
[2] `https://www.dbpedia.org/resources/latest-core/`

data across decentralised repositories and heterogeneous RDF engines. Federated query processors have been improved in many aspects, including querying documents, query planner optimisation and sensitive data access policies [17–23]. A federated query execution framework depends on source selection as it determines which endpoints are relevant to evaluate a given query. The Source selection is a separate step before execution, aiming to reduce the number of requests to the RDF server, optimise resource allocations and minimise network overhead, improving overall query execution time.

Well-specified communication is crucial for distributed RDF engines and federated queries, leading to significant performance increases. For communication-focused programming, a framework of Multiparty Session Types (MPST) provides a typed process that abstracts communication between multiple participants into a global protocol (type) [24]. The theory formalise endpoint projection of the global description in the Web Services Choreography Description Language[3], but it can accommodate other diverse specifications [25]. We aim to integrate this typed theory with SPARQL. This theory matches common issues in the software development process and is essential for a collaborative team of engineers writing SPARQL. A global type (protocol) gives engineers a global view of federated query communication between RDF repositories. The engineers can validate their SPARQL federated query following a global protocol.

We highlight current SPARQL issues as follows. (1) *Source discovery*: Writing a SPARQL query that involves many RDF repositories and complex query joins can be burdensome; and (2) *Performance and completeness*: Query execution time depends on the performance of SPARQL runtime and federated communication cost. More importantly, the query must return a complete result.

This paper introduces a novel SPARQL Query Template (SQT) generator that leverages the benefits of Multiparty Session Types [24, 26, 27]. The theory notably gives some clarity to complex processes. The SQT generates federated query body syntax that includes the RDF URL source address and guarantees conformance with the projection of a given global type. Moreover, by using a global type to describe communication patterns, we can identify and prevent overhead communications, speed up query execution and ensure the completeness of the results.

In the remainder, Section 2 gives preliminaries of RDF, MPST and SPARQL relational algebra through examples. Section 3 illustrates a use case that motivates this work. We explain in detail how SQT produces query templates from a global type. Section 4 explains how local types conform to relational algebra. Section 5 evaluates different network communication costs associated with different implementations. Finally, Section 6 discusses current limitations and future works of current work.

[3] `https://www.w3.org/2002/ws/chor/`

2 Preliminaries and Examples

This section introduces RDF notations, SPARQL relational algebra and MPST through simple examples.

2.1 RDF and SPARQL

Our basic definitions of RDF and SPARQL follow [28, 29]. In addition, we add a representation of RDF graph [16] and conjunctive query [7].

Definition 1 (RDF Triple). *RDF terms* (t) *are consist of IRIs* ($\mathbf{I}$)*, literals* ($\mathbf{L}$) *and blank nodes* ($\mathbf{B}$)*. These terms form a triple or 3-tuple,* $\langle t_s, t_p, t_o\rangle \in (\mathbf{I} \cup \mathbf{B}) \times (\mathbf{I}) \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$*, where* s*,* p *and* o *denote subject, predicate and object of the triple,* $\beta = \{s, p, o\}$*.*

A set of RDF triples indicates a semantic relationship between terms and can be represented as a directed graph [16]. We can construct and label multiple graphs into a dataset. However, for convenience, this paper focuses on a single graph in a dataset.

Definition 2 (RDF Dataset). *Dataset (D) is a set of triples. The semantic relationship on triples can be described as a directed labelled graph. A graph consists of vertices* (V) *and directed edges* (E)*, denoted by* $D = \{V, E\}$*. Vertices* V *denotes a set of resource nodes* $t_s \cup t_o \in V$*. Edges* E *is a set of directed edges connecting the nodes in* V*. To simplify, the directed edge* $t_s \xrightarrow{t_p} t_o$ *is expressed by* $\langle t_s, t_p, t_o\rangle$*.*

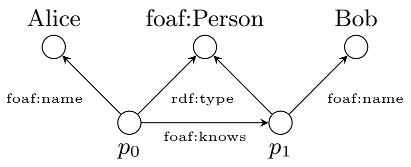


Fig. 1: RDF Dataset

Example 1. To illustrate, we define a dataset carrying a relation between Alice and Bob, presented as vertices and directed edges as shown in Fig. 1.

$$
\begin{aligned}
D =& \{V, E\}\\
V =& \{\text{Alice}, \text{Bob}, \text{foaf:Person}, p_0, p_1\}\\
E =& \{\langle p_0, \text{foaf:knows}, p_1\rangle, \langle p_0, \text{rdf:type}, \text{foaf:Person}\rangle,\\
& \langle p_1, \text{rdf:type}, \text{foaf:Person}\rangle, \langle p_0, \text{foaf:name}, \text{Alice}\rangle, \langle p_1, \text{foaf:name}, \text{Bob}\rangle\}
\end{aligned}
$$

Definition 3 (SPARQL Query Template). *The SPARQL Query Template (SQT) syntax is denoted as follows:*

$$\begin{aligned} Q, Q^{'} ::=\ & A && \textit{(triple pattern or atom)} \\ \mid\ & \mathcal{S}.Q^{'} && \textit{(service federation)} \\ \mid\ & Q \wedge Q^{'} && \textit{(group of basic graph pattern)} \end{aligned}$$

The SPARQL relational algebra formally defines a query language semantics [7,28,29]. A query's result or solution is a tuple containing variables from queries matched against the RDF dataset. This paper recursively constructs queries from graph patterns, including triple patterns, basic graph patterns and service federation.

Definition 4 (Triple Pattern or Atom). *Triple pattern or atom* (A) *forms* $\langle t_s, t_p, t_o\rangle \in (\mathbf{I}\cup\mathbf{B}\cup\mathbf{V}) \times (\mathbf{I}\cup\mathbf{V}) \times (\mathbf{I}\cup\mathbf{B}\cup\mathbf{L}\cup\mathbf{V})$ *where* $(\mathbf{V})$ *refers to variables. The variables occurring in an atom are denoted by* $vars(A)$*. The triple pattern evaluates RDF triples in* D *by selection function* (σ) *with constant terms* t_k *and by mapping function* (π) *that maps terms* t_j *to the variables* v_i*:*

$$A ::= \pi_{v_i \leftarrow t_j}(\sigma_{t_k}(D)) \qquad (\ j,k \in \beta \quad \text{and} \quad i \in \mathbb{N}\ )$$

$$vars(A) = \{t_\beta | t_\beta \cap \mathbf{V}\} \qquad (\ t_k \in (\mathbf{I}\cup\mathbf{B}\cup\mathbf{L}) \quad \text{and} \quad t_j \in (\mathbf{I}\cup\mathbf{B}\cup\mathbf{L}\cup\mathbf{V})\ )$$

Example 2. A query for finding a person's name on dataset D is defined as:

```
{?x foaf:name ?y}
```

This query translates to the relational operation:

$$\langle x, \text{foaf:name}, y\rangle = \pi_{x,y \leftarrow t_s,t_o}(\sigma_{t_p=\text{foaf:name}}(D))$$

The process begins with searching any triples in a dataset that match the selection's predicate, $t_p = \text{foaf:name} \in \mathbf{I}$. Here, we have two variables `?x` and `?y` in subject and object respectively. Subject $t_s \in \mathbf{I}\cup\mathbf{B}\cup\mathbf{V}$ and object $t_o \in \mathbf{I}\cup\mathbf{B}\cup\mathbf{L}\cup\mathbf{V}$ intersect with $\mathbf{V}$ and will be included in $vars(A) = \{t_s, t_o\}$.

Definition 5 (Basic Graph Pattern). *The basic graph pattern is a group of triple patterns. It can be expressed syntactically as a conjunctive query Q from finite set atoms, where vars(Q) denotes variables occurring in the basic graph pattern.*

$$Q ::= A_1 \wedge \cdots \wedge A_n \qquad \text{and} \qquad vars(Q) = vars(A_1) \cup \cdots \cup vars(A_n)$$

Multiple triple patterns stacked together will construct a basic graph pattern [7]. The basic graph pattern variables are the union of all triple pattern variables. These graph patterns can be evaluated in a local or remote dataset, formalised in Definition 6 [29]. The definition of set variables also applies equally to remote datasets.

Definition 6 (Service Federation). *The SPARQL service ($\mathcal{S}$) is a query evaluation on a remote dataset. Query evaluation Q on remote service $\mathcal{S}$ is denoted as ($\mathcal{S}.Q$) while local query evaluation is denoted as (Q).*

The SPARQL query Q is evaluated over dataset D and then returns solution mapping that represents matched variables on triples in a dataset [29]. The solution mapping function is defined in Def. 7. We assume all variables are bounded and no error returns from the graph pattern.

Definition 7 (Solution Mapping Function). *A solution mapping (π) partially maps variables $\mathbf{V}$ from terms $\mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$. Suppose $\{v_i, \ldots, v_n\} \subseteq \mathbf{V}$ and $\{x_i, \ldots, x_n\} \subseteq vars(Q)$ for $n \in \mathbb{N}$. Then we denote by $\{v_i/x_i, \ldots, v_n/x_n\}$ mapping function π such that the solution contains a set of $dom(\phi) = \{v_i|x_i\}$.*

$$\pi_{v_i} = x_i \quad (v_i \in dom(\pi)) \qquad \textit{and} \qquad \pi_{\emptyset} = \emptyset \quad (dom(\pi) = \emptyset)$$

Example 3. Assume there are two datasets. The local dataset contains triples of persons. The remote dataset contains people's relations with each other. The first graph pattern is evaluated locally and the second pattern is service federation, which is evaluated on a remote dataset as shown in Fig. 2. The local graph pattern finds subject terms with type foaf:Person and map into variable `?x` $\in \mathbf{I}$. Then, a service federation pattern finds the person `?y` $\in \mathbf{I}$ that is acquainted with `?x`. A complete query syntax is presented in Fig. 3.

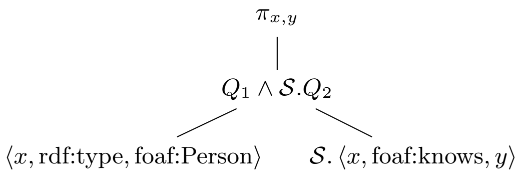

$\pi_{x,y}$
$Q_1 \wedge \mathcal{S}.Q_2$
$\langle x, \text{rdf:type}, \text{foaf:Person} \rangle$
$\mathcal{S}.\langle x, \text{foaf:knows}, y \rangle$

Fig. 2: SPARQL relational algebra

```
SELECT ?x ?y /* return variables */
WHERE {
  ?x rdf:type foaf:Person . /* graph pattern is evaluated on local dataset */
  SERVICE <https://remote>{
    ?x foaf:knows ?y . /* graph pattern is evaluated on remote dataset */
  }
}
```

Fig. 3: SPARQL 1.1 Query

2.2 Multiparty Session Types

Global types (protocol) G provides a blueprint describing communication between participants as a pair of a sender and a receiver [24, 26, 27]. In Definition 8, *message type* formalises protocol where `p` sends to `q` a value with type U. Similarly, *branching type* formalises participant `p` chooses a label l_i from `q` for some $i \in I$. The *termination type* **end** denotes communication is finished and no further process.

Definition 8 (Global and Local Types). *The global type G and local type T are denoted as follows:*

$$\begin{array}{lll} G ::= \mathbf{end} \;\big|\; \mathtt{p} \to \mathtt{q} : [U]\,; G \;\big|\; \mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I} & & \textit{(termination, message, branching)} \\ T ::= \mathbf{end} \;\big|\; \mathtt{q}!\,[U]; T \;\big|\; \mathtt{p}?\,[U]; T & & \textit{(termination, send, receive)} \\ \quad \big|\; \mathtt{q}{\oplus}\{l_i : T_i\}_{i \in I} \;\big|\; \mathtt{p}\&\{l_i : T_i\}_{i \in I} & & \textit{(internal choice, external choice)} \end{array}$$

We require that $\mathtt{p} \neq \mathtt{q}$, $I \neq \emptyset$, and $i \neq j$ whenever $i \neq j$, for all $i, j \in I$.

Local type T defines a process for a single participant following a global type G [24, 26, 27]. *Termination type* ($\mathbf{end}$) denotes the end of a process. *Send type* $\mathtt{q}![U]; T$, sends a message to $\mathtt{q}$ with payload U. *Receive type* $\mathtt{p}![U]; T$, waits for a message from $\mathtt{p}$ with payload U. *External choice* $\mathtt{p}\&\{l_i : T_i\}_{i \in I}$ waits for selection label l_i from local type to $\mathtt{p}$ to continues. *Internal choice* $\mathtt{q}{\oplus}\{l_i : T_i\}_{i \in I}$ formalises a label selection l_i at $\mathtt{q}$ and continues.

Definition 9 (Participants). *We define the set of participants of a global type G as follows:*

$$\begin{aligned} pt(\mathbf{end}) &= \emptyset \\ pt(\mathtt{p} \to \mathtt{q} : [U]\,; G) &= \{\mathtt{p}, \mathtt{q}\} \cup pt(G) \\ pt(\mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I}\,; G) &= \{\mathtt{p}, \mathtt{q}\} \cup \textstyle\bigcup_{i \in I} pt(G_i) \end{aligned}$$

A set of participants in *message type* is their sender $\mathtt{p}$ and receiver $\mathtt{q}$ which are defined in a global type. Similarly, for *branching type*, a set of participants is the label sender $\mathtt{p}$ and label receiver $\mathtt{q}$. A *branching type* has multiple continuations G_i and the set of participants is a union from all branches. A *termination type* does not have continuation and does not involve any participants.

Definition 10 (Projection). *The projection of a global type G onto a participant $\mathtt{r}$ is defined as follows:*

$$\begin{aligned} \mathbf{end} \upharpoonright \mathtt{r} &= \mathbf{end} \\ \mathtt{p} \to \mathtt{q} : [U]\,; G \upharpoonright \mathtt{r} &= \begin{cases} \mathtt{q}![U]; G \upharpoonright \mathtt{r} & \mathtt{r} = \mathtt{p} \\ \mathtt{p}?[U]; G \upharpoonright \mathtt{r} & \mathtt{r} = \mathtt{q} \\ G \upharpoonright \mathtt{r} & \mathtt{r} \notin \{\mathtt{p}, \mathtt{q}\} \end{cases} \\ \mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I}\,; G \upharpoonright \mathtt{r} &= \begin{cases} \mathtt{q}{\oplus}\{l_i : G_i {\upharpoonright} \mathtt{r}\}_{i \in I} & \mathtt{r} = \mathtt{p} \\ \mathtt{p}\&\{l_i : G_i {\upharpoonright} \mathtt{r}\}_{i \in I} & \mathtt{r} = \mathtt{q} \\ \sqcap_{i \in I} G_i \upharpoonright \mathtt{r} & \mathtt{r} \notin \{\mathtt{p}, \mathtt{q}\} \end{cases} \end{aligned}$$

A global type can be projected into a local type of participant $G \upharpoonright \mathtt{r} = T_{\mathtt{r}}$. The *termination type* $\mathbf{end}$ does not transform during the projection. The projection of *message type* becomes a *send type* where $\mathtt{r} = \mathtt{p}$, $\mathtt{r}$ is a sender and becomes a *receive type* where $\mathtt{r} = \mathtt{q}$, $\mathtt{r}$ is a receiver. The *branching type* is projected to an *external choice* where $\mathtt{r} = \mathtt{p}$ and into an *internal choice* where $\mathtt{r} = \mathtt{q}$. Projection of *branching type* to the third participant $\mathtt{r}$, neither $\mathtt{p}$ nor $\mathtt{q}$, must specified to have the same continuation for any branch label.

Definition 11 (Global Type Reduction). *Evaluating* $\mathtt{p} \to \mathtt{q}$ *in global type* G *is defined as follows:*

(message reduction)

$$(\mathtt{p} \to \mathtt{q} : [U] \,; G) \setminus \mathtt{p} \xrightarrow{U} \mathtt{q} = G$$

(pass message reduction)

$$(\mathtt{r} \to \mathtt{s} : [U] \,; G) \setminus \mathtt{p} \xrightarrow{U'} \mathtt{q} = \mathtt{r} \to \mathtt{s} : [U]; (G \setminus \mathtt{p} \xrightarrow{U'} \mathtt{q})$$
$$\textit{if } \{\mathtt{r}, \mathtt{s}\} \cap \{\mathtt{p}, \mathtt{q}\} = \emptyset \textit{ and } \{\mathtt{p}, \mathtt{q}\} \subseteq G$$

(branching reduction)

$$(\mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I}) \setminus \mathtt{p} \xrightarrow{l} \mathtt{q} = G_i \qquad \textit{for } l = l_i$$

(pass branching reduction)

$$(\mathtt{r} \to \mathtt{s} : \{l_i : G_i\}_{i \in I}) \setminus \mathtt{p} \xrightarrow{l} \mathtt{q} = \mathtt{r} \to \mathtt{s} : \left\{ l_i : G_i \setminus \mathtt{p} \xrightarrow{l} \mathtt{q} \right\}_{i \in I}$$
$$\textit{if } \{\mathtt{r}, \mathtt{s}\} \cap \{\mathtt{p}, \mathtt{q}\} = \emptyset \textit{ and } \forall i \in I : \{\mathtt{p}, \mathtt{q}\} \subseteq G_i$$

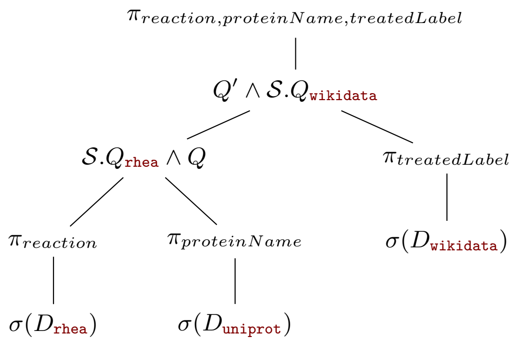
Message reduction $\mathtt{p} \xrightarrow{U} \mathtt{q}$ reduces the global type G with *message type* $\mathtt{p} \to \mathtt{q}$: $[U]$ and continues to the next protocol G. *Branching reduction* $\mathtt{p} \xrightarrow{l} \mathtt{q}$ reduces multiple label selection $\mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I}$ by choosing one label l_i then continue with G_i. *Pass reduction* applies when participants of reduction $\{\mathtt{p}, \mathtt{q}\}$ do not meet $\{\mathtt{r}, \mathtt{s}\}$ such as reduction $\mathtt{p} \xrightarrow{U} \mathtt{q}$ on $\mathtt{r} \to \mathtt{s}$: $[U]$ or $\mathtt{p} \xrightarrow{l} \mathtt{q}$ on $\mathtt{r} \to \mathtt{s} : \{l_i : G_i\}_{i \in I}$. The reduction continues through the rest of the global type until it meets the same participant.

3 Overview

Many research institutes [1–6] provide open RDF datasets. These datasets are often used towards drug discovery. Researchers use them to work with genomic, metabolomic, molecular structures, reaction pathways and screening drug candidates. The information resides in different domains and locations. Data integration and retrieval are the keys to comprehensive analyses. The SPARQL federated query techniques allow engineers to query remote datasets like a single unified database.

We use UniProt query example number 45[4] to cover all discussions in this paper. The use case describes a query of retrieving drug targets for human enzymes involved in sterol metabolism. The example uses three SPARQL endpoints, Rhea[5], UniProt[6] and Wikidata[7]. The query can be run directly at UniProt query editor. Alternatively, it can be executed in Rhea or Wikidata but requires

[4] `https://sparql.uniprot.org/.well-known/sparql-examples/?offset=40`
[5] `https://sparql.rhea-db.org/sparql`
[6] `https://sparql.uniprot.org/sparql`
[7] `https://query.wikidata.org/sparql`


$\pi_{reaction,proteinName,treatedLabel}$
$Q' \wedge \mathcal{S}.Q_{\texttt{wikidata}}$
$\mathcal{S}.Q_{\texttt{rhea}} \wedge Q$
$\pi_{treatedLabel}$
$\pi_{reaction}$
$\pi_{proteinName}$
$\sigma(D_{\texttt{wikidata}})$
$\sigma(D_{\texttt{rhea}})$
$\sigma(D_{\texttt{uniprot}})$

$$Q_{\texttt{uniprot}} = \mathcal{S}.Q_{\texttt{rhea}} \wedge Q_{\texttt{local}} \wedge \mathcal{S}.Q_{\texttt{wikidata}}$$

$$T_{\texttt{uniprot}} = \texttt{rhea}?[U]; \texttt{local}?[U]; \texttt{wikidata}?[U]; \textbf{end}$$

Fig. 4: The relational algebra of retrieving drug targets for human enzymes involved in sterol metabolism

some rewriting at the federated query part. We aim to make it less tiring to write a federated query by modelling this in a global type.

We present a query relational algebra and its local type representation in Fig. 4. The query involves three participants: `rhea`, `uniprot` and `wikidata`. Query $Q_{\texttt{uniprot}}$ is evaluated at `uniprot`. Conjunctive query $Q_{\texttt{uniprot}}$ consists of two service federations and a local graph pattern. The query evaluation follows the sequence: Rhea, UniProt and Wikidata. First, `uniprot` evaluates federated query at `rhea` and retrieves reactions involving some molecules. Then, it joins the result with a local query at `uniprot` to find human enzymes that catalyse reactions. Finally, `uniprot` finds the interaction of enzymes from previous results with drug descriptions in `wikidata`. It is important to note that query $Q_{\texttt{uniprot}}$ and local type $T_{\texttt{uniprot}}$ are specifications for `uniprot` only. Other participants will need a different query algebra and a local type definition. The global type provides a global view and ensures accurate and consistent results across all participants during query execution.

4 Implementation of SQT Workflow

This section explains the workflow used to generate the SPARQL Query Template. The top-down diagram in Fig. 5 shows the process from global type to query template. We define global type G with four participants: `rhea`, `uniprot`, `wikidata` and `local`. The `local` participant physically does not exist. Instead, `local` acts as `rhea`, `uniprot` or `wikidata`.

We begin with building a global type G, in Example 4, using the Scribble script defined in Fig. 6. Suppose a script `Payload() from p to q;` the participant `q` expects a message return from `p`. This is defined as $\texttt{p} \to \texttt{q} : [U]$ where payload U contains free tuples of terms t_β. Here, `p` is a participant who owns a dataset and executes query Q either in a local or remote environment.

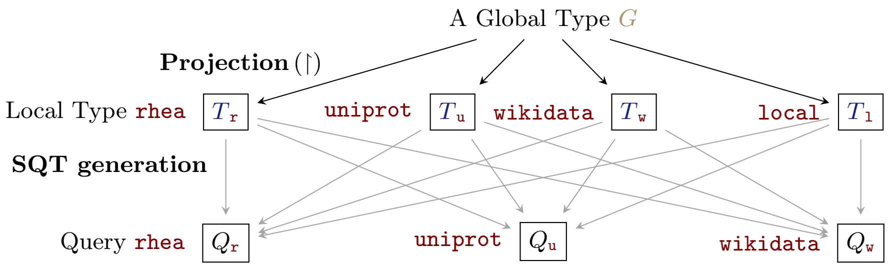


Fig. 5: Top-down view of global type for UniProt query number 45. The `r` stand for `rhea`, `u` for `uniprot`, `w` for `wikidata` and `l` stand for `local`

The global type is implemented in Scribble script in Fig. 6. Lines 4-8 represents $G_{\texttt{rhea}}$ a choice of `local` acts as `rhea`, lines 18-22, $G_{\texttt{uniprot}}$ where `local` as `uniprot` and lines 32-36, $G_{\texttt{wikidata}}$ where `local` as `wikidata`. Implementation of `rhea` evaluates query locally shown in line 10 and federated query in line 11. The definition of global type G is required to be projectable. Otherwise, it cannot produce any local types T and queries Q.

We introduce a *broadcast type* with the global type definition. The *broadcast type* formed as a sequence of *branching type* and uses one label that same as the previously selected branch. In Definition 12, participant $\mathtt{p}$ broadcast a l_i to $\bar{\mathtt{q}}$ is expressed by selecting a label l_i to a finite set number of participant $\bar{\mathtt{q}} = \{\mathtt{q}_1, \mathtt{q}_2, \ldots \mathtt{q}_n\}$.

Definition 12 (Broadcast Type). *A broadcast type is sending or selecting the same label to multiple participants. A complete set of global type syntax with broadcast type is given as follows:*

$$G ::= \textbf{end} \mid \mathtt{p} \to \bar{\mathtt{q}} : [U] ; G \qquad \textit{(termination, message)}$$
$$\mid \mathtt{p} \to \bar{\mathtt{q}} : \{l_i : G_i\}_{i \in I} \qquad \textit{(branching)}$$

$$\mathtt{p} \to \bar{\mathtt{q}} : [U] ; G \overset{def}{=} \mathtt{p} \to \mathtt{q}_1 : [U] ; \mathtt{p} \to \mathtt{q}_1 : [U] ; \ldots$$
$$\mathtt{p} \to \mathtt{q}_n : [U] ; G \qquad \textit{(broadcast message)}$$

$$\mathtt{p} \to \bar{\mathtt{q}} : \{l_i : G_i\}_{i \in I} \overset{def}{=} \mathtt{p} \to \mathtt{q}_1 : \{l_i : \mathtt{p} \to \mathtt{q}_2 : \{l_i : \ldots$$
$$\mathtt{p} \to \mathtt{q}_n : \{l_i : G_i\}\}\}_{i \in I} \qquad \textit{(broadcast branching)}$$

$$\textit{with } \bar{\mathtt{q}} = \{\mathtt{q}_1, \mathtt{q}_2, \ldots, \mathtt{q}_n\} \qquad n \in \mathbb{N}$$

The broadcast reduction evaluates communication between participants of a *broadcast type* in global type G or continues to go through the next types when participants involved in broadcast type differ from the reduction criteria. Reducing *branching type* $\mathtt{p} \xrightarrow{l_i} \mathtt{q}_1$ or $\mathtt{p} \xrightarrow{l_i} \mathtt{q}_2$ in global type G can happen individually in a different order, but a broadcast reduction must reduce all $\bar{\mathtt{q}}$ together.

Definition 13 (Broadcast Reduction). *Evaluating a broadcast type in a global type G is defined as follows:*

(message reduction)

$$(\mathtt{p} \to \bar{\mathtt{q}} : [U] \,; G) \setminus \mathtt{p} \xrightarrow{U} \bar{\mathtt{q}} \cong G$$

(pass message reduction)

$$(\mathtt{r} \to \bar{\mathtt{s}} : [U] \,; G) \setminus \mathtt{p} \xrightarrow{U'} \bar{\mathtt{q}} \cong \mathtt{r} \to \bar{\mathtt{s}} : [U]; (G \setminus \mathtt{p} \xrightarrow{U'} \bar{\mathtt{q}})$$
$$\textit{if } \{\mathtt{r}, \bar{\mathtt{s}}\} \cap \{\mathtt{p}, \bar{\mathtt{q}}\} = \emptyset \textit{ and } \{\mathtt{p}, \bar{\mathtt{q}}\} \subseteq G$$

(branching reduction)

$$\mathtt{p} \to \bar{\mathtt{q}} : \{l_i : G_i\}_{i \in I} \setminus \mathtt{p} \xrightarrow{l} \bar{\mathtt{q}} \cong G_i \qquad l = l_i$$

(pass branching reduction)

$$\mathtt{r} \to \bar{\mathtt{s}} : \{l_i : G_i\}_{i \in I} \setminus \mathtt{p} \xrightarrow{l} \bar{\mathtt{q}} \cong \mathtt{r} \to \bar{\mathtt{s}} : \left\{l_i : G_i \setminus \mathtt{p} \xrightarrow{l} \bar{\mathtt{q}}\right\}_{i \in I}$$
$$\textit{if } \{\mathtt{r}, \bar{\mathtt{s}}\} \cap \{\mathtt{p}, \bar{\mathtt{q}}\} = \emptyset \textit{ and } \forall i \in I : \{\mathtt{p}, \bar{\mathtt{q}}\} \subseteq G_i$$

Theorem 1 (Broadcast). *If* $G \setminus \mathtt{p} \xrightarrow{l} \bar{\mathtt{q}} \cong G'$ *then* $G \setminus \mathtt{p} \xrightarrow{l} \mathtt{q}_1 \setminus \mathtt{p} \xrightarrow{l} \mathtt{q}_2 \cdots \setminus \mathtt{p} \xrightarrow{l} \mathtt{q}_n = G'$ *where* $n \in \mathbb{N}$ *and* $\bar{\mathtt{q}} = \{\mathtt{q}_1, \mathtt{q}_2, \ldots, \mathtt{q}_n\}$.

Proof. By induction on n and G.

Example 4. We formalise a global type for UniProt query number 45 as follows:

$$I = \{\mathtt{rhea}, \mathtt{uniprot}, \mathtt{wikidata}\}$$
$$G = \mathtt{local} \to \mathtt{rhea}\{l_i : \mathtt{local} \to \mathtt{uniprot}\{l_i : \mathtt{local} \to \mathtt{wikidata}\{l_i : G_i\}\}\}_{i \in I}$$

$$\begin{aligned}
G_{\mathtt{rhea}} &= \mathtt{local} \to \mathtt{rhea} : [U]; \mathtt{uniprot} \to \mathtt{rhea} : [U]; \\
&\quad \mathtt{wikidata} \to \mathtt{rhea} : [U]; \mathbf{end} \\
G_{\mathtt{uniprot}} &= \mathtt{rhea} \to \mathtt{uniprot} : [U]; \mathtt{local} \to \mathtt{uniprot} : [U]; \\
&\quad \mathtt{wikidata} \to \mathtt{uniprot} : [U]; \mathbf{end} \\
G_{\mathtt{wikidata}} &= \mathtt{rhea} \to \mathtt{wikidata} : [U]; \mathtt{uniprot} \to \mathtt{wikidata} : [U]; \\
&\quad \mathtt{local} \to \mathtt{wikidata} : [U]; \mathbf{end}
\end{aligned}$$

Below is a global type reduction for `local` by choosing label $l_i = l$:

$$G \setminus \mathtt{local} \xrightarrow{l} \{\mathtt{rhea}, \mathtt{uniprot}, \mathtt{wikidata}\} = G_i$$

Algorithm 1 translates a global type to a query template, following translation on Table 1. The algorithm has two inputs, the projection of global protocol $G \upharpoonright \mathtt{p}$ and target participant $\mathtt{q}$ where a query will be executed. Suppose we use a global type G in Example 4 to build a query template for `uniprot`. This global type G has three branches for different continuations: $G_{\mathtt{rhea}}$, $G_{\mathtt{uniprot}}$ and $G_{\mathtt{wikidata}}$. Initially, we set $\mathtt{p} = \mathtt{uniprot}$ for $G \upharpoonright \mathtt{p}$ and participant $\mathtt{q} = \mathtt{uniprot}$.

```
global protocol FederatedQuery(role Local, role Rhea, role UniProt,role Wikidata){
    choice at Local{
        /* Local tells everyone that it chooses to act as Rhea */
        Rhea() from Local to Rhea;
        choice at Local{
            Rhea() from Local to UniProt;
            choice at Local{
                Rhea() from Local to Wikidata;
                /* Rhea */
                Select(reaction) from Local to Rhea;
                Select(protein,proteinFullName) from UniProt to Rhea;
                Select(chemical,chemicalLabel,treatedLabel) from Wikidata to Rhea;
            }
        }
    }
    or {
        /* Local tells everyone that it chooses to act as UniProt */
        UniProt() from Local to Rhea;
        choice at Local{
            UniProt() from Local to UniProt;
            choice at Local{
                UniProt() from Local to Wikidata;
                /* UniProt */
                Select(reaction) from Rhea to UniProt;
                Select(protein,proteinFullName) from Local to UniProt;
                Select(chemical,chemicalLabel,treatedLabel) from Wikidata to UniProt;
            }
        }
    }
    or {
        /* Local tells everyone that it chooses to act as Wikidata */
        Wikidata() from Local to Rhea;
        choice at Local{
            Wikidata() from Local to UniProt;
            choice at Local{
                Wikidata() from Local to Wikidata;
                /* Wikidata */
                Select(reaction) from Rhea to Wikidata;
                Select(protein,proteinFullName) from UniProt to Wikidata;
                Select(chemical,chemicalLabel,treatedLabel) from Local to Wikidata;
            }
        }
    }
}
```

Fig. 6: Scribble script of global type for UniProt query number 45

Next, global type G reduced by selecting a branch for $\texttt{uniprot}$, $\backslash\texttt{local} \xrightarrow{l} \bar{\texttt{s}}$ where $l = \texttt{q}$ and $\bar{\texttt{s}} = \{\texttt{rhea}, \texttt{uniprot}, \texttt{wikidata}\}$. Selecting label $l_i = l$ continues with G_i shows the role of $\texttt{local}$ which can either become $\texttt{rhea}$ following with $G_{\texttt{rhea}}$, $\texttt{uniprot}$ with $G_{\texttt{uniprot}}$, or $\texttt{wikidata}$ with $G_{\texttt{wikidata}}$.

The Algorithm 1 at line number 2 evaluates $G_i \upharpoonright \texttt{p}$ whether it is an empty (**end**) or continues to be translated. In the case of $\texttt{p} = \texttt{q}$, a *message type* in $G_i \upharpoonright \texttt{p} = \texttt{r}?[U]$ and $\texttt{r}$ is a $\texttt{local}$ then translates a message type to a query Q. This query will be executed locally at $\texttt{p}$. On the other hand, the message type is translated to service federation $\mathcal{S}_{\texttt{p}}.Q$, when $\texttt{p}$ is neither $\texttt{local}$ or current target participant $\texttt{q}$. In summary, $G_{\texttt{uniprot}} \upharpoonright \texttt{uniprot}$ translates to a local query in $\texttt{uniprot}$ and $G_{\texttt{uniprot}} \upharpoonright \texttt{rhea}$ explains that $\texttt{uniprot}$ sends and evaluates a query to $\texttt{rhea}$.

Table 1: Translation of local type and SPARQL relational algebra

Local type $(T_{\mathtt{q}} = G \upharpoonright \mathtt{q})$		**SPARQL algebra** (Q_q)
$[\![\mathbf{end}]\!]$	$=$	$\emptyset$
$[\![\mathtt{p}?\,[U]\,;G \upharpoonright \mathtt{q}]\!]\quad \mathtt{p} \neq \mathit{Local}$	$=$	$\mathcal{S}_p.[\![G \upharpoonright \mathtt{p}]\!] \wedge [\![G \upharpoonright \mathtt{q}]\!]$
$[\![\mathtt{p}?\,[U]\,;G \upharpoonright \mathtt{q}]\!]\quad \mathtt{p} = \mathit{Local}$	$=$	$Q_{\mathtt{q}} \wedge [\![G \upharpoonright \mathtt{q}]\!]$
$[\![\mathtt{p}\,!\,[U]\,;G \upharpoonright \mathtt{q}]\!]$	$=$	$[\![G \upharpoonright \mathtt{q}]\!]$
$[\![\mathtt{p}\oplus\{l_i : G_i \upharpoonright \mathtt{q}\}_{i\in I}]\!]$	$=$	$[\![G_i \upharpoonright \mathtt{q}]\!]_{i\in I}$
$[\![\mathtt{p}\&\{l_i : G_i \upharpoonright \mathtt{q}\}_{i\in I}]\!]$	$=$	$[\![G_i \upharpoonright \mathtt{q}]\!]_{i\in I}$

Table 1 only considers a *message type* to be translated directly into a query. The **end** type is translated to nothing and represents the end of a query. However, *internal choice* and *external choice*, defined in Table 1, do not have a query translation. Instead, it continues to translate the following local types. Although it does not have query translation, it plays an essential part in the reduction process by deciding what is `local` participant role will be.

In a simple federation query where `uniprot` calls federation service to both `rhea` and `wikidata`. Here, `rhea` and `wikidata` send their result directly to `uniprot` and do not expect to receive a result from calling another federation service. Both local types $T_{\mathtt{rhea}}$ and $T_{\mathtt{wikidata}}$ define $\mathtt{rhea}![U]$ which does not translate to any queries. The algorithm continues to read recursively and reduces the local type from all participants until it is finished.

In a nested service federation, each local type for three participants, including `rhea`, `uniprot` and `wikidata` may have called a federation service to other participants. For example, `wikidata` expects a federation service result from `uniprot`,

Algorithm 1 Construct SPARQL Query Template

```
1:  function SQT($G \upharpoonright \mathtt{p}, \mathtt{q}$)            ▷ begin with $p = q$ where $\mathtt{p}, \mathtt{q} \in pt(G) \setminus \{\mathtt{local}\}$
2:      while $(G \setminus \mathtt{local} \xrightarrow{l} \bar{\mathtt{s}}) \upharpoonright \mathtt{p} \neq \mathbf{end}; l = \mathtt{q}$ do      ▷ $\bar{\mathtt{s}} = pt(G) \setminus \{\mathtt{local}\}$
3:          if $T_{\mathtt{p}} = \mathtt{r}?[U]$ then                          ▷ $\mathtt{p} \neq \mathtt{r}; \mathtt{r} \in pt(G)$
4:              if $\mathtt{r} = \mathtt{local}$ then
5:                  $Q_{\mathtt{q}} \leftarrow [\![\mathtt{r}?[U]]\!]$                          ▷ local query $Q$
6:              else
7:                  $\mathcal{S}_{\mathtt{r}}.(Q_{\mathtt{q}}) \leftarrow$ SQT($G \upharpoonright \mathtt{r}, \mathtt{q}$)              ▷ federated query $\mathcal{S}.Q$
8:              end if
9:          end if
10:         $Q'_{\mathtt{q}} = Q_{\mathtt{q}} \wedge \mathcal{S}_{\mathtt{r}}.Q_{\mathtt{q}} \wedge \cdots \wedge \mathcal{S}_{\mathtt{r}'}.Q_{\mathtt{q}}$                  ▷ Conjunctive query
11:     end while
12:     return $Q'_{\mathtt{q}}$
13: end function
```

$T_{\texttt{wikidata}} = \texttt{uniprot}?[U]$. Meanwhile, uniprot is waiting federation service result from rhea before it can be sent to wikidata, $T_{\texttt{uniprot}} = \texttt{rhea}?[U]; \texttt{wikidata}![U]$. The SQT algorithm requires definitions from all participant's local types in order to generate a complete query template. The algorithm begins by generating a local query and service federation for target participant q then calling a related local type to generate a query in federated service.

5 Evaluation

We conduct experiments in the docker environment. We build a small-scale environment with three Apache Fuseki containers and a triple dataset to imitate Rhea, UniProt and Wikidata. For statistics rhea contains 2051 triples, uniprot 129902 triples and wikidata 6592 triples. We record the elapsed projection time, constructing SQT and performance differences between variants of federated query communication that follow the current use case.

Example 5. We formalise the general structure of a global protocol (type) for a current evaluation as follows:

$$G \setminus \texttt{local} \xrightarrow{l} \bar{\texttt{q}} = G' \qquad \textit{where} \qquad l \in pt(G) \setminus \{\texttt{local}\}$$

We prepare global type G following Example 5. We assign varying numbers of participants to global type G: 10, 30, and 60. Table 2 shows the elapsed time for constructing SQT. The projection column shows the time needed to generate all local types. It shows that increasing the number of participants will result in a longer elapsed time.

We prepared three global types following Example 5 for service federation performance. We define each global type to have three specifications for rhea, uniprot and wikidata. We have nine different test cases in total. Global type G_1 calls two remote service federations from one participant, following Example 4. However, global types G_2 and G_3 call nested service federations. Global type G_2 is given in Example 6.

Example 6. Based on the global type in Example 5, we define $G_{\texttt{uniprot}}$ such that uniprot can only call rhea service federation through wikidata. The complete

Table 2: Construction time for single query

	Time (second)	
Number of participants in G	**Projection** (total)	**SQT** (average)
10 participants	0.445	0.321
30 participants	2.720	0.724
60 participants	30.624	1.429

continuation of global type G_i for nested service federation defined as follows:

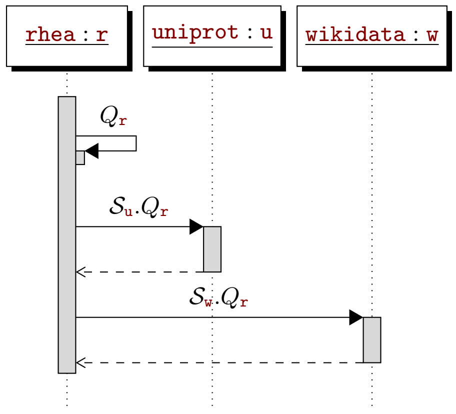
$$\begin{aligned}
I &= \{\mathtt{rhea}, \mathtt{uniprot}, \mathtt{wikidata}\}\\
G_{\mathtt{rhea}} &= \mathtt{local} \to \mathtt{rhea} : [U]; \mathtt{uniprot} \to \mathtt{rhea} : [U];\\
&\quad \mathtt{wikidata} \to \mathtt{uniprot} : [U]; \mathbf{end}\\
G_{\mathtt{uniprot}} &= \mathtt{rhea} \to \mathtt{wikidata} : [U]; \mathtt{wikidata} \to \mathtt{uniprot} : [U];\\
&\quad \mathtt{local} \to \mathtt{uniprot} : [U]; \mathtt{wikidata} \to \mathtt{uniprot} : [U]; \mathbf{end}\\
G_{\mathtt{wikidata}} &= \mathtt{rhea} \to \mathtt{wikidata} : [U]; \mathtt{uniprot} \to \mathtt{rhea} : [U];\\
&\quad \mathtt{local} \to \mathtt{wikidata} : [U]; \mathbf{end}
\end{aligned}$$

We define a simplified notation in Table 3, `l` for `local`, `r` for `rhea`, `u` for `uniprot` and `w` for `wikidata`. We show federated query elapsed time with different communication sequences. Suppose we take the first case in Table 3 which is generating a query template for `rhea`, $Q'_{\mathtt{r}} = Q_{\mathtt{r}} \wedge \mathcal{S}_{\mathtt{u}}.Q_{\mathtt{r}} \wedge \mathcal{S}_{\mathtt{w}}.Q_{\mathtt{r}}$. Based on Algorithm 1 and an input data $[\![(G_1 \setminus \mathtt{l} \xrightarrow{\mathtt{r}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \restriction \mathtt{r}]\!]$, global type G_1 is reduced by telling every participant that `local` will acts as `rhea` under reduction of $G_1 \setminus \mathtt{local} \xrightarrow{\mathtt{rhea}} \{\mathtt{rhea}, \mathtt{uniprot}, \mathtt{wikidata}\}$. This syntactically will produce $G_{\mathtt{rhea}}$. Global type is projected to `rhea`, $G_{\mathtt{rhea}} \restriction \mathtt{rhea} = T_{\mathtt{rhea}}$, before being translated into a query with Table 1.

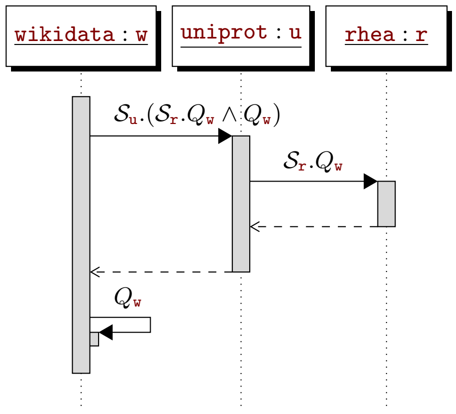
The first query template in Table 3 has an overhead communication exchange between participants, as shown in Fig. 7a and local type definitions. Based on the query execution sequence in Fig. 4, a result from `rhea` must join with `uniprot` to retrieve reactions and related human enzymes. Next, a result must join with `wikidata` to find interactions between those previous enzymes and drug descriptions. Ideally, `uniprot` sends the join result directly to `wikidata` but local type $T_{\mathtt{u}} = \mathtt{r}![U];\mathbf{end}$ defines a communication that `uniprot` can send only to `rhea`. The communication `uniprot` to `wikidata` passes through `rhea` first and contin-

Table 3: Query elapsed time from different global protocol

Query Template	**Elapsed time** (second)
1. $[\![(G_1 \setminus \mathtt{l} \xrightarrow{\mathtt{r}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \restriction \mathtt{r}]\!] = Q_{\mathtt{r}} \wedge \mathcal{S}_{\mathtt{u}}.Q_{\mathtt{r}} \wedge \mathcal{S}_{\mathtt{w}}.Q_{\mathtt{r}}$	23.160
2. $[\![(G_1 \setminus \mathtt{l} \xrightarrow{\mathtt{u}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \restriction \mathtt{u}]\!] = \mathcal{S}_{\mathtt{r}}.Q_{\mathtt{u}} \wedge Q_{\mathtt{u}} \wedge \mathcal{S}_{\mathtt{w}}.Q_{\mathtt{u}}$	12.255
3. $[\![(G_1 \setminus \mathtt{l} \xrightarrow{\mathtt{w}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \restriction \mathtt{w}]\!] = \mathcal{S}_{\mathtt{r}}.Q_{\mathtt{w}} \wedge \mathcal{S}_{\mathtt{u}}.Q_{\mathtt{w}} \wedge Q_{\mathtt{w}}$	11.334
4. $[\![(G_2 \setminus \mathtt{l} \xrightarrow{\mathtt{r}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \restriction \mathtt{r}]\!] = Q_{\mathtt{r}} \wedge \mathcal{S}_{\mathtt{u}}(Q_{\mathtt{r}} \wedge \mathcal{S}_{\mathtt{w}}.Q_{\mathtt{r}})$	24.500
5. $[\![(G_2 \setminus \mathtt{l} \xrightarrow{\mathtt{u}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \restriction \mathtt{u}]\!] = \mathcal{S}_{\mathtt{w}}(\mathcal{S}_{\mathtt{r}}.Q_{\mathtt{u}}) \wedge Q_{\mathtt{u}} \wedge \mathcal{S}_{\mathtt{w}}.Q_{\mathtt{u}}$	14.824
6. $[\![(G_2 \setminus \mathtt{l} \xrightarrow{\mathtt{w}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \restriction \mathtt{w}]\!] = \mathcal{S}_{\mathtt{r}}(Q_{\mathtt{w}} \wedge \mathcal{S}_{\mathtt{u}}.Q_{\mathtt{w}}) \wedge Q_{\mathtt{w}}$	13.076
7. $[\![(G_3 \setminus \mathtt{l} \xrightarrow{\mathtt{r}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \restriction \mathtt{r}]\!] = Q_{\mathtt{r}} \wedge \mathcal{S}_{\mathtt{w}}(\mathcal{S}_{\mathtt{u}}.Q_{\mathtt{r}} \wedge Q_{\mathtt{r}})$	23.084
8. $[\![(G_3 \setminus \mathtt{l} \xrightarrow{\mathtt{u}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \restriction \mathtt{u}]\!] = \mathcal{S}_{\mathtt{r}}.Q_{\mathtt{u}} \wedge Q_{\mathtt{u}} \wedge \mathcal{S}_{\mathtt{r}}(\mathcal{S}_{\mathtt{w}}.Q_{\mathtt{u}})$	19.786
9. $[\![(G_3 \setminus \mathtt{l} \xrightarrow{\mathtt{w}} \mathtt{r}, \mathtt{u}, \mathtt{w}) \restriction \mathtt{w}]\!] = \mathcal{S}_{\mathtt{u}}(\mathcal{S}_{\mathtt{r}}.Q_{\mathtt{w}} \wedge Q_{\mathtt{w}}) \wedge Q_{\mathtt{w}}$	4.241


rhea : r
uniprot : u
wikidata : w
$Q_{\mathtt{r}}$
$\mathcal{S}_{\mathtt{u}}.Q_{\mathtt{r}}$
$\mathcal{S}_{\mathtt{w}}.Q_{\mathtt{r}}$

(a) Case 1: `local` (`l`) act as `rhea`
$T_{\mathtt{r}} = \mathtt{l}?[U]; \mathtt{u}?[U]; \mathtt{w}?[U];\mathbf{end}$
$T_{\mathtt{u}} = \mathtt{r}![U];\mathbf{end}$
$T_{\mathtt{w}} = \mathtt{r}![U];\mathbf{end}$


wikidata : w
uniprot : u
rhea : r
$\mathcal{S}_{\mathtt{u}}.(\mathcal{S}_{\mathtt{r}}.Q_{\mathtt{w}} \wedge Q_{\mathtt{w}})$
$\mathcal{S}_{\mathtt{r}}.Q_{\mathtt{w}}$
$Q_{\mathtt{w}}$

(b) Case 9 : `local` (`l`) act as `wikidata`
$T_{\mathtt{r}} = \mathtt{u}![U];\mathbf{end}$
$T_{\mathtt{u}} = \mathtt{r}?[U]; \mathtt{w}![U];\mathbf{end}$
$T_{\mathtt{w}} = \mathtt{u}?[U]; \mathtt{l}?[U];\mathbf{end}$

Fig. 7: Federated service implementation between two global protocols

ues to `wikidata` together with federated query $\mathcal{S}_{\mathtt{w}}.Q_{\mathtt{r}}$. Finally, `wikidata` sends result back to `rhea`. This causes a longer execution time because `uniprot` cannot send directly to `wikidata` and must pass `rhea` first.

In contrast, case number 9 makes the most efficient communication as shown in Fig. 7b, as the join result is sent directly to the next target participant without an intermediary. According to local type $T_{\mathtt{r}} = \mathtt{u}![U];\mathbf{end}$, `rhea` sends directly to `uniprot`. Next, `uniprot` sends a query result to `wikidata` as a final destination, $T_{\mathtt{u}} = \mathtt{r}?[U]; \mathtt{w}![U];\mathbf{end}$.

6 Conclusion

The current SQT (SPARQL Query Template) algorithm is a convenient and faster way of generating queries than writing them manually. The use of global types helps identify RDF sources by checking the interaction between participants. Global types ensure that executing queries on all projections will provide a complete result set. Moreover, the communication sequence described in global types can also be used to check for any overhead communication during a federated query. These benefits can significantly reduce network costs and ultimately make federated queries faster. In addition, we introduce a broadcast reduction for global types and prove its soundness against the original semantics of global types.

Acknowledgements. The first author is funded by the LPDP Indonesian Endowment Fund for Education from the Ministry of Finance of the Republic of Indonesia. This research is funded in part by EPSRC EP/T006544/2, EP/K011715/1, EP/K034413/1, EP/L00058X/1, EP/N027833/2, EP/N028201/1, EP/T014709/2, EP/V000462/1, EP/X015955/1, NCSS/EPSRC VeTSS and Horizon EU TaRDIS 101093006.

References

1. Mendes, P.N., Jakob, M., García-Silva, A., Bizer, C.: DBpedia spotlight: shedding light on the web of documents. In: Proceedings of the 7th International Conference on Semantic Systems. p. 1–8. I-Semantics '11, Association for Computing Machinery, New York, NY, USA (2011), https://doi.org/10.1145/2063518.2063519
2. Bateman, A., et al.: UniProt: the Universal Protein Knowledgebase in 2023. Nucleic Acids Research 51(D1), D523–D531 (jan 2023)
3. Bansal, P., Morgat, A., Axelsen, K.B., Muthukrishnan, V., Coudert, E., Aimo, L., Hyka-Nouspikel, N., Gasteiger, E., Kerhornou, A., Neto, T.B., Pozzato, M., Blatter, M.C., Ignatchenko, A., Redaschi, N., Bridge, A.: Rhea, the reaction knowledgebase in 2022. Nucleic Acids Research 50(D1), D693–D700 (jan 2022), https://academic.oup.com/nar/article/50/D1/D693/6424769
4. Aleksander, S.A., et al.: The Gene Ontology knowledgebase in 2023. GENETICS 224(1) (may 2023), https://academic.oup.com/genetics/article/doi/10.1093/genetics/iyad031/7068118
5. Bauer-Mehren, A., Bundschus, M., Rautschka, M., Mayer, M.A., Sanz, F., Furlong, L.I.: Gene-Disease Network Analysis Reveals Functional Modules in Mendelian, Complex and Environmental Diseases. PLoS ONE 6(6), e20284 (jun 2011), https://dx.plos.org/10.1371/journal.pone.0020284
6. Mendez, D., Gaulton, A., Bento, A.P., Chambers, J., De Veij, M., Félix, E., Magariños, M.P., Mosquera, J.F., Mutowo, P., Nowotka, M., Gordillo-Marañón, M., Hunter, F., Junco, L., Mugumbate, G., Rodriguez-Lopez, M., Atkinson, F., Bosc, N., Radoux, C.J., Segura-Cabrera, A., Hersey, A., Leach, A.R.: ChEMBL: towards direct deposition of bioassay data. Nucleic Acids Research 47(D1), D930–D940 (jan 2019), https://academic.oup.com/nar/article/47/D1/D930/5162468
7. Potter, A., Motik, B., Nenov, Y., Horrocks, I.: Distributed RDF query answering with dynamic data exchange. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 9981 LNCS, 480–497 (2016)
8. Sidirourgos, L., Goncalves, R., Kersten, M., Nes, N., Manegold, S.: Column-store support for RDF data management. Proceedings of the VLDB Endowment 1(2), 1553–1563 (aug 2008), https://dl.acm.org/doi/10.14778/1454159.1454227
9. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: SW-Store: a vertically partitioned DBMS for Semantic Web data management. The VLDB Journal 18(2), 385–406 (apr 2009), http://link.springer.com/10.1007/s00778-008-0125-y
10. Atre, M., Chaoji, V., Zaki, M.J., Hendler, J.A.: Matrix "Bit" loaded. In: Proceedings of the 19th international conference on World wide web. pp. 41–50. ACM, New York, NY, USA (apr 2010), https://dl.acm.org/doi/10.1145/1772690.1772696
11. Neumann, T., Weikum, G.: The RDF-3X engine for scalable management of RDF data. The VLDB Journal 19(1), 91–113 (feb 2010), http://link.springer.com/10.1007/s00778-009-0165-y
12. Rohloff, K., Schantz, R.E.: Clause-Iteration with MapReduce to scalably query data graphs in the SHARD graph-store. In: DIDC'11 - Proceedings of the 4th International Workshop on Data-Intensive Distributed Computing. pp. 35–44. ACM (2011)

13. Zhang, X., Chen, L., Tong, Y., Wang, M.: EAGRE: Towards scalable I/O efficient SPARQL query evaluation on the cloud. In: Proceedings - International Conference on Data Engineering. pp. 565–576. IEEE (2013)
14. Hassan, M., Bansal, S.: S3QLRDF: distributed SPARQL query processing using Apache Spark—a comparative performance study, vol. 41. Springer US (2023), https://doi.org/10.1007/s10619-023-07422-4
15. Zeng, K., Yang, J., Wang, H., Shao, B., Wang, Z.: A distributed graph engine for web scale RDF data. Proceedings of the VLDB Endowment 6(4), 265–276 (2013)
16. Gurajada, S., Seufert, S., Miliaraki, I., Theobald, M.: TriAD: A distributed shared-nothing RDF engine based on asynchronous message passing. Proceedings of the ACM SIGMOD International Conference on Management of Data pp. 289–300 (2014)
17. Daga, E., Asprino, L., Mulholland, P., Gangemi, A.: Facade-X: An Opinionated Approach to SPARQL Anything. In: Alam, M., Groth, P., de Boer, V., Pellegrini, T., Pandit, H.J. (eds.) Volume 53: Further with Knowledge Graphs, vol. 53, pp. 58–73. IOS Press (August 2021), http://oro.open.ac.uk/78973/
18. Hernawan, A., Sunarwidhi, A., Prasedya, E., Widyastuti, S.: A generalization SPARQL federated query: An initial step towards machine-readable web of data for halal food products. IOP Conference Series: Earth and Environmental Science 913(1), 012040 (nov 2021), https://iopscience.iop.org/article/10.1088/1755-1315/913/1/012040
19. Quilitz, B., Leser, U.: Querying distributed RDF data sources with SPARQL. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 5021 LNCS, 524–538 (2008)
20. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: A federation layer for distributed query processing on linked open data. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Lecture Notes in Computer Science, vol. 6643 LNCS, pp. 481–486. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
21. Khan, Y., Saleem, M., Mehdi, M., Hogan, A., Mehmood, Q., Rebholz-Schuhmann, D., Sahay, R.: SAFE: SPARQL Federation over RDF Data Cubes with Access Control. Journal of Biomedical Semantics 8(1), 5 (dec 2017), http://jbiomedsem.biomedcentral.com/articles/10.1186/s13326-017-0112-6
22. Görlitz, O., Staab, S.: SPLENDID: SPARQL endpoint federation exploiting VOID descriptions. In: Proceedings of the Second International Conference on Consuming Linked Data - Volume 782. p. 13–24. COLD'11, CEUR-WS.org, Aachen, DEU (2011)
23. Troumpoukis, A., Charalambidis, A., Mouchakis, G., Konstantopoulos, S., Siebes, R., de Boer, V., Soiland-Reyes, S., Digles, D.: Developing a Benchmark Suite for Semantic Web Data from Existing Workflows. In: Fundulaki, I., Krithara, A., Ngomo, A.N., Rentoumi, V. (eds.) Proceedings of the Workshop on Benchmarking Linked Data (BLINK 2016) co-located with the 15th International Semantic Web Conference (ISWC), Kobe, Japan, October 18, 2016. CEUR Workshop Proceedings, vol. 1700. CEUR-WS.org (2016), https://ceur-ws.org/Vol-1700/paper-04.pdf
24. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. Journal of the ACM 63(1), 1–67 (2016)
25. Yoshida, N.: Programming Language Implementations with Multiparty Session Types. In: de Boer, F.S., Damiani, F., Hähnle, R., Johnsen, E.B., Kamburjan, E. (eds.) Active Object Languages: Current Research Trends, Lecture Notes in Computer Science, vol. 14360, pp. 147–165. Springer (2024), https://doi.org/10.1007/978-3-031-51060-1_6

26. Coppo, M., Dezani-Ciancaglini, M., Padovani, L., Yoshida, N.: A gentle introduction to multiparty asynchronous session types. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 9104, 146–178 (2015)
27. Yoshida, N., Gheri, L.: A Very Gentle Introduction to Multiparty Session Types. In: 16th International Conference on Distributed Computing and Internet Technology, LNCS, vol. 11969, pp. 73–93. Springer (2020)
28. Cyganiak, R.: A relational algebra for SPARQL. HP, `http://www.hpl.hp.com/techreports/2005/HPL-2005-170.pdf` (2005)
29. Salas, J., Hogan, A.: Semantics and canonicalisation of SPARQL 1.1. Semantic Web 13(5), 829–893 (2022)