FFI Forsvarets forskningsinstitutt

FFI Forsvarets forskningsinstitutt

Supporting Evacuation Missions with Ontology-based SPARQL Federation

Audun Stolpe and Jonas Halvorsen

STIDS 12-15 November 2013

Table of contents

The NNEC Concept

- Official policy for Command and Control in NATO forces
- Goal: Achieve better mission effectiveness
- Builds upon extensive information sharing
- Shift from “Need to know” to “Responsibiliy to share”

NNEC objectives

NNEC objectives

- Some technical aims:

NNEC objectives

- Some technical aims:
  - support extensive information sharing,

NNEC objectives

- Some technical aims:
  - support extensive information sharing,
  - provide a robust scheme for information integration,

NNEC objectives

- Some technical aims:
  - support extensive information sharing,
  - provide a robust scheme for information integration,
  - standardise exchange and storage formats

NNEC objectives

- Some technical aims:
  - support extensive information sharing,
  - provide a robust scheme for information integration,
  - standardise exchange and storage formats
- Some strategic aims:

NNEC objectives

- Some technical aims:
  - support extensive information sharing,
  - provide a robust scheme for information integration,
  - standardise exchange and storage formats
- Some strategic aims:
  - create a high degree of shared situational awareness,

NNEC objectives

- Some technical aims:
  - support extensive information sharing,
  - provide a robust scheme for information integration,
  - standardise exchange and storage formats
- Some strategic aims:
  - create a high degree of shared situational awareness,
  - based on timely data,

NNEC objectives

- Some technical aims:
  - support extensive information sharing,
  - provide a robust scheme for information integration,
  - standardise exchange and storage formats
- Some strategic aims:
  - create a high degree of shared situational awareness,
  - based on timely data,
  - to support decision making during operations

Presuppositions

Presuppositions

- NNEC assumptions:

Presuppositions

- NNEC assumptions:
  - fragile ICT infrastructure, typically IP radio

Presuppositions

- NNEC assumptions:
  - fragile ICT infrastructure, typically IP radio
  - low bandwidth + latency issues

Presuppositions

- NNEC assumptions:
  - fragile ICT infrastructure, typically IP radio
  - low bandwidth + latency issues
  - information is typically distributed across systems

Presuppositions

- NNEC assumptions:
  - fragile ICT infrastructure, typically IP radio
  - low bandwidth + latency issues
  - information is typically distributed across systems
  - systems are contributed by different coalition members

Presuppositions

- NNEC assumptions:
  - fragile ICT infrastructure, typically IP radio
  - low bandwidth + latency issues
  - information is typically distributed across systems
  - systems are contributed by different coalition members
  - systems may appear and disappear

Presuppositions

- NNEC assumptions:
  - fragile ICT infrastructure, typically IP radio
  - low bandwidth + latency issues
  - information is typically distributed across systems
  - systems are contributed by different coalition members
  - systems may appear and disappear
  - information may be mission critical

Presuppositions

- NNEC assumptions:
  - fragile ICT infrastructure, typically IP radio
  - low bandwidth + latency issues
  - information is typically distributed across systems
  - systems are contributed by different coalition members
  - systems may appear and disappear
  - information may be mission critical
- Our assumptions:

Presuppositions

- NNEC assumptions:
  - fragile ICT infrastructure, typically IP radio
  - low bandwidth + latency issues
  - information is typically distributed across systems
  - systems are contributed by different coalition members
  - systems may appear and disappear
  - information may be mission critical
- Our assumptions:
  - data models are standardised

Presuppositions

- NNEC assumptions:
  - fragile ICT infrastructure, typically IP radio
  - low bandwidth + latency issues
  - information is typically distributed across systems
  - systems are contributed by different coalition members
  - systems may appear and disappear
  - information may be mission critical
- Our assumptions:
  - data models are standardised
  - HTTP can be assumed

Desiderata

Desiderata

We want an approach that:

Desiderata

We want an approach that:

1. allows a user to access available sources in a unified way,

Desiderata

We want an approach that:

1. allows a user to access available sources in a unified way,
2. minimises the number of required HTTP requests

Desiderata

We want an approach that:

1. allows a user to access available sources in a unified way,
2. minimises the number of required HTTP requests
3. allows the relevant sources to be discovered at run-time

Desiderata

We want an approach that:

1. allows a user to access available sources in a unified way,
2. minimises the number of required HTTP requests
3. allows the relevant sources to be discovered at run-time
4. guarantees the soundness/completeness of q. a.

Desiderata

We want an approach that:

1. allows a user to access available sources in a unified way,
2. minimises the number of required HTTP requests
3. allows the relevant sources to be discovered at run-time
4. guarantees the soundness/completeness of q. a.

Desiderata

We want an approach that:

1. allows a user to access available sources in a unified way,
2. minimises the number of required HTTP requests
3. allows the relevant sources to be discovered at run-time
4. guarantees the soundness/completeness of q. a.

- Points to ontology based data access + rewriting

Table of contents

Case description

- JOC — Joint Operations Center

Case description

- JOC — Joint Operations Center
- analyst is monitoring medical evacuation missions

Case description

- JOC — Joint Operations Center
- analyst is monitoring medical evacuation missions
- particularly missions that are threatened by enemy activity

Case description

- JOC — Joint Operations Center
- analyst is monitoring medical evacuation missions
- particularly missions that are threatened by enemy activity
- information need expressed by the query

Case description

- JOC — Joint Operations Center
- analyst is monitoring medical evacuation missions
- particularly missions that are threatened by enemy activity
- information need expressed by the query

"Find all medical evacuation missions and friendly units such that a) the mission can be classified as being threatened; and b) that the friendly unit can handle the specific type of threat that the enemy poses."

Information sources

Information sources

The query involves three operational information systems:

Information sources

The query involves three operational information systems:

1. `JOCWatch`: a log of events reported from the field

Information sources

The query involves three operational information systems:

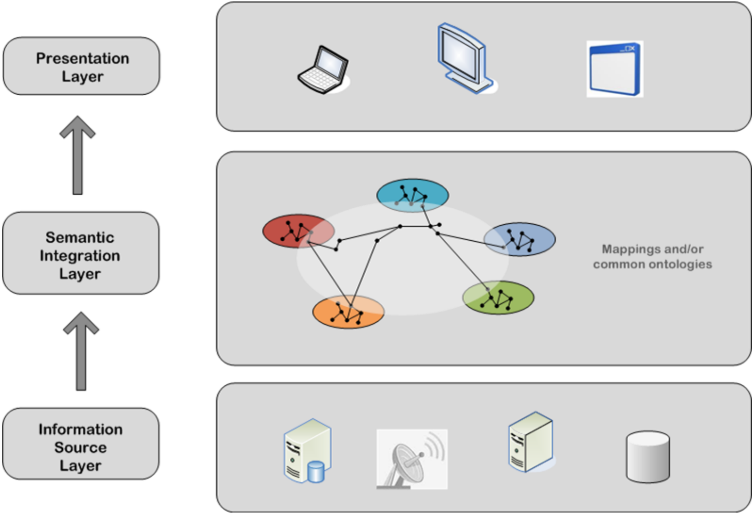
1. `JOCWatch`: a log of events reported from the field
2. `MedWatch`: medical mission planning and tracking

Information sources

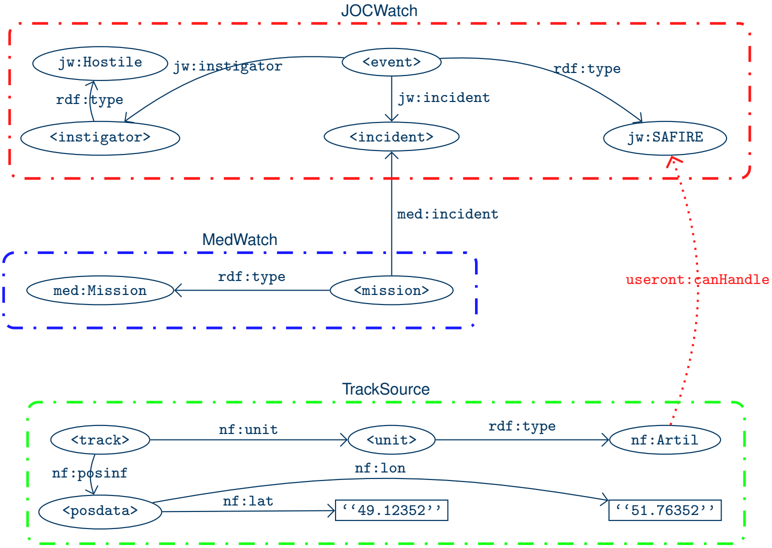
The query involves three operational information systems:

1. `JOCWatch`: a log of events reported from the field
2. `MedWatch`: medical mission planning and tracking
3. `Track Source`: friendly units: capabilities, location.

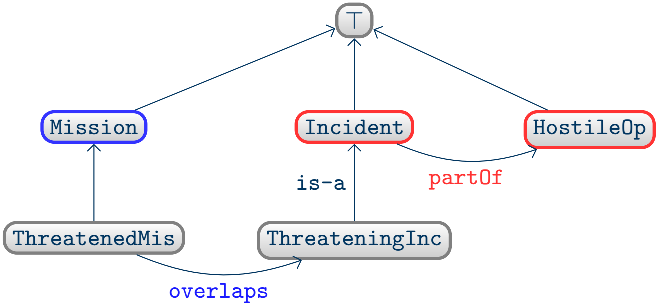
Ontology-based data integration



Conceptual relationship between data sources

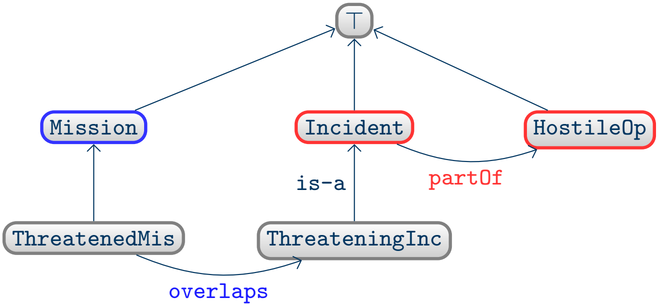


Rewriting: A simple example



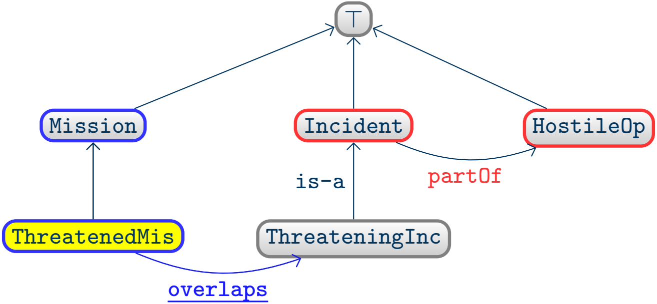
Rewriting: A simple example

Query: `ThreatenedMission(x)?`



Rewriting: A simple example

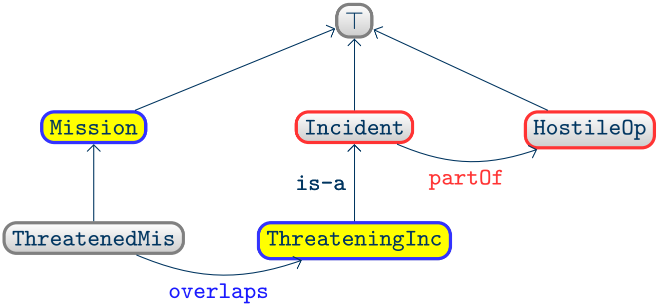
Query: `ThreatenedMission(x)?`



`overlaps(x, y) &`

Rewriting: A simple example

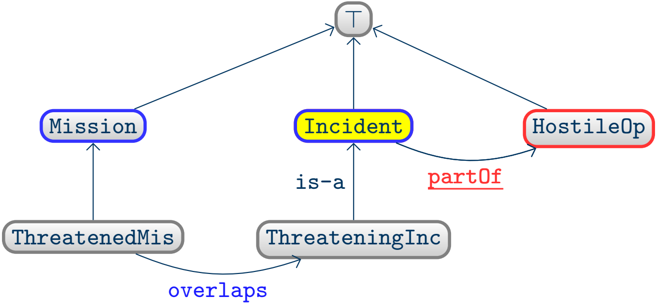
Query: ThreatenedMission(x)?



Mission(x) & overlaps(x, y) &

Rewriting: A simple example

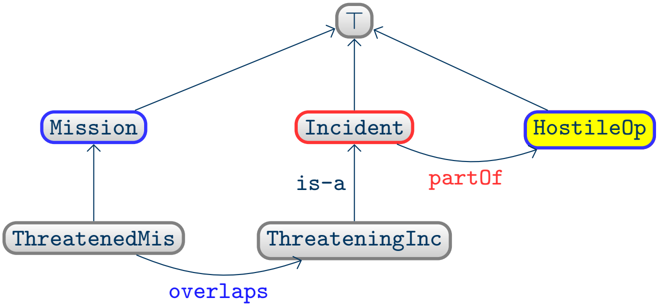
Query: `ThreatenedMission(x)?`



`Mission(x) & overlaps(x, y) & Incident(y) & partOf(y,z) &`

Rewriting: A simple example

Query: ThreatenedMission(x)?



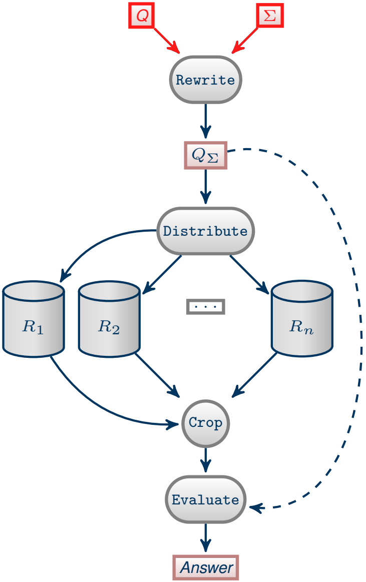
Mission(x) & overlaps(x, y) & Incident(y) & partOf(y,z) &
HostileOp(z)

Table of contents

System overview

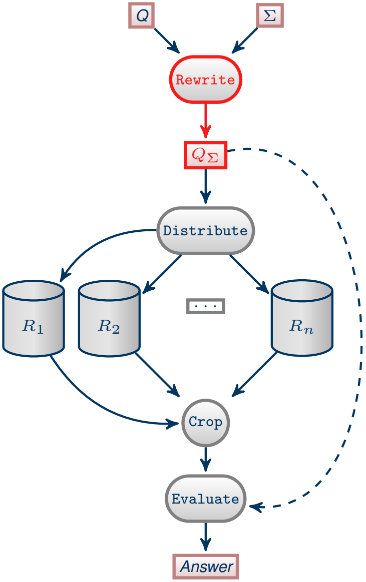
System overview

- Input: SPARQL query Q and an ontology Σ



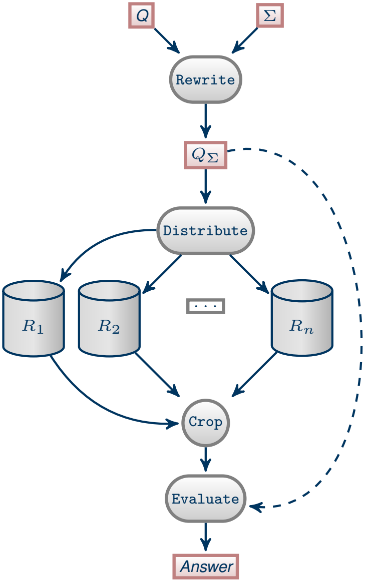
System overview

- Input: SPARQL query Q and an ontology Σ
- Q rewritten to Q_Σ



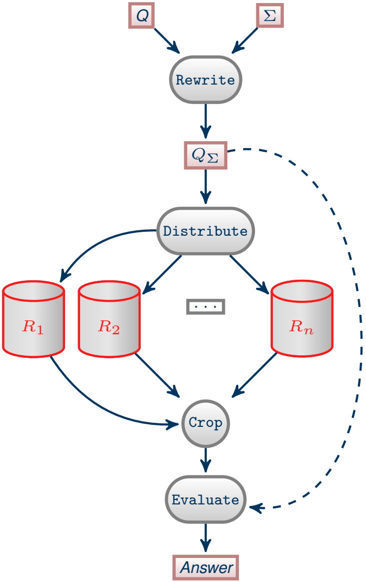
System overview

- Input: SPARQL query Q and an ontology Σ
- Q rewritten to Q_Σ
- federator selects from available sources



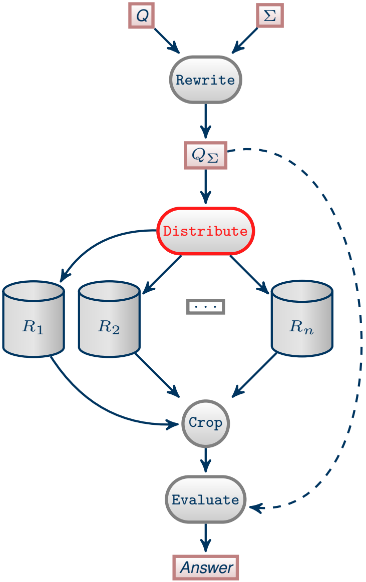
System overview

- Input: SPARQL query Q and an ontology Σ
- Q rewritten to Q_Σ
- federator selects from available sources
- according to signature of Q_Σ, then



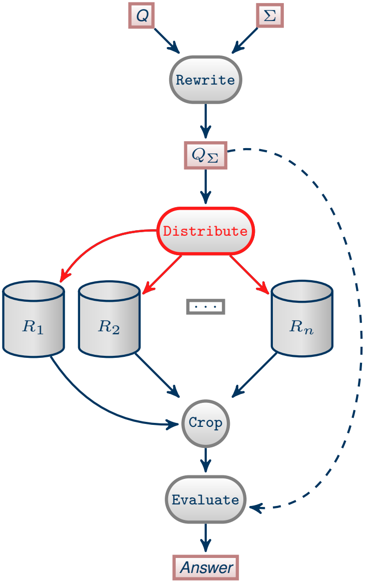
System overview

- Input: SPARQL query Q and an ontology Σ
- Q rewritten to Q_Σ
- federator selects from available sources
- according to signature of Q_Σ, then
- splits Q into subqueries tailored for each source, and



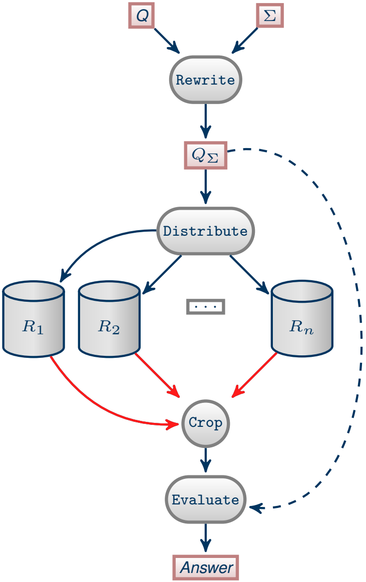
System overview

- Input: SPARQL query Q and an ontology Σ
- Q rewritten to Q_Σ
- federator selects from available sources
- according to signature of Q_Σ, then
- splits Q into subqueries tailored for each source, and
- distributes subqueries to the sources



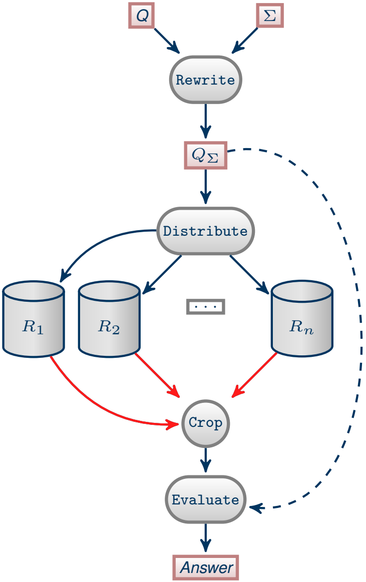
System overview

- Input: SPARQL query Q and an ontology Σ
- Q rewritten to Q_Σ
- federator selects from available sources
- according to signature of Q_Σ, then
- splits Q into subqueries tailored for each source, and
- distributes subqueries to the sources
- results of subqueries yield snapshots, that



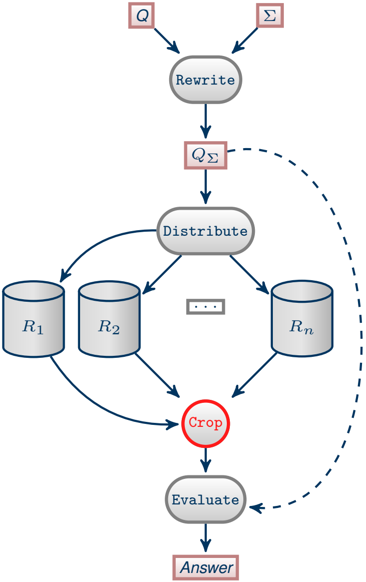
System overview

- Input: SPARQL query Q and an ontology Σ
- Q rewritten to Q_Σ
- federator selects from available sources
- according to signature of Q_Σ, then
- splits Q into subqueries tailored for each source, and
- distributes subqueries to the sources
- results of subqueries yield snapshots, that
- are jointly sufficient for answering Q



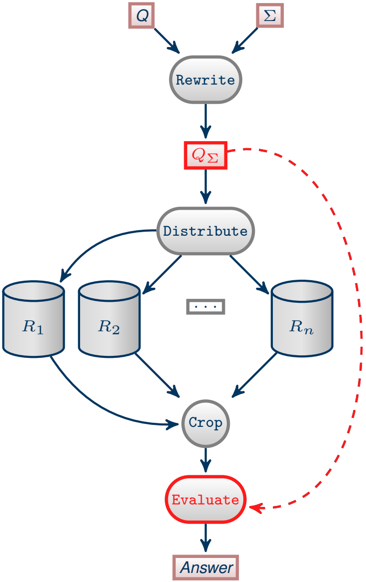
System overview

- Input: SPARQL query Q and an ontology Σ
- Q rewritten to Q_Σ
- federator selects from available sources
- according to signature of Q_Σ, then
- splits Q into subqueries tailored for each source, and
- distributes subqueries to the sources
- results of subqueries yield snapshots, that
- are jointly sufficient for answering Q
- snapshots are joined to form the *cropping* $\mathcal{A}$



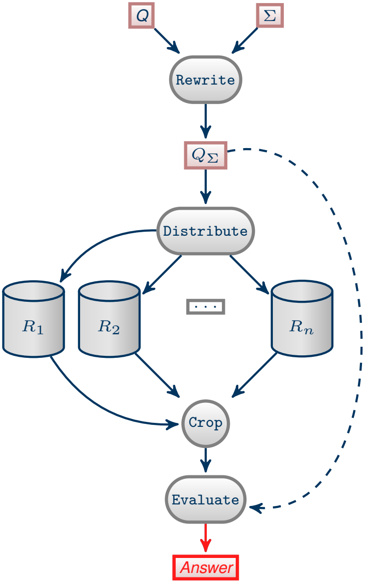
System overview

- Input: SPARQL query Q and an ontology Σ
- Q rewritten to Q_Σ
- federator selects from available sources
- according to signature of Q_Σ, then
- splits Q into subqueries tailored for each source, and
- distributes subqueries to the sources
- results of subqueries yield snapshots, that
- are jointly sufficient for answering Q
- snapshots are joined to form the *cropping* $\mathcal{A}$
- against which, Q_Σ is finally evaluated



System overview

- Input: SPARQL query Q and an ontology Σ
- Q rewritten to Q_Σ
- federator selects from available sources
- according to signature of Q_Σ, then
- splits Q into subqueries tailored for each source, and
- distributes subqueries to the sources
- results of subqueries yield snapshots, that
- are jointly sufficient for answering Q
- snapshots are joined to form the *cropping* $\mathcal{A}$
- against which, Q_Σ is finally evaluated



Sample query

Query pattern:

```
?mission medics:missionType medics:Rescue.
?mission medics:jocWatchIncident ?incident.
?incident jocw:status ?stat.
```

Routed to `MedWatch`:

```
CONSTRUCT {
 ?_1 medics:missionType medics:Evac.
 ?_1 medics:jocwIncident ?_2.
 ?_3 jocw:status ?_4.
 } WHERE {
 { ?_1 medics:missionType medics:Evac.
   ?_1 medics:jocwIncident ?_2.}
 UNION
 { ?_3 jocw:status ?_4.}}
```

Routed to `JOCWatch`:

```
CONSTRUCT {
 ?_1 jocw:instigator ?_2.
 ?_3 jocw:status ?_4.
 } WHERE {
 { ?_1 jocw:instigator ?_2.}
 UNION
 { ?_3 jocw:status ?_4.}}
```

Each query is built as a union of exclusive/nonexclusive groups

Soundness/completeness

- Note the renaming of variables in the queries
- necessary for the *soundness* of query answering
- `CONSTRUCT` queries are defined to adhere to a logical form
- guarantees soundness/completeness in the sense:

> *Distributing a query Q over a set of sources R yields the exact same answer, as if Q were evaluated directly against a single repository containing the union of R.*

Table of contents

Adapting to a dynamic network topology

- Important design goals:
  - do not hard-wire a query to a predefined set of sources
  - sources should be selected once per query
  - as sources come and go, the federator adapts

- Facilitated by FOL-rewritable ontology languages
  - FOL-rewritability decouples reasoning from data access
  - queries are precompiled (potentially cached)
  - selection and distribution performed afterwards
  - thus, network topology allowed to change

Detecting available source

- Discovery mechanism

Detecting available source

- Discovery mechanism
    - mDNS for broadcasting and discovering sources,

Detecting available source

- Discovery mechanism
  - mDNS for broadcasting and discovering sources,
  - DNS-SD content description and endpoint location

Detecting available source

- Discovery mechanism
  - mDNS for broadcasting and discovering sources,
  - DNS-SD content description and endpoint location
  - SPARQL 1.1 SDs and VoID descriptions of source content.

Detecting available source

- Discovery mechanism
  - mDNS for broadcasting and discovering sources,
  - DNS-SD content description and endpoint location
  - SPARQL 1.1 SDs and VoID descriptions of source content.
- Advantages:

Detecting available source

- Discovery mechanism
  - mDNS for broadcasting and discovering sources,
  - DNS-SD content description and endpoint location
  - SPARQL 1.1 SDs and VoID descriptions of source content.
- Advantages:
  - approach addresses the NNEC needs

Detecting available source

- Discovery mechanism
  - mDNS for broadcasting and discovering sources,
  - DNS-SD content description and endpoint location
  - SPARQL 1.1 SDs and VoID descriptions of source content.
- Advantages:
  - approach addresses the NNEC needs
  - it is independent of a central registry,

Detecting available source

- Discovery mechanism
  - mDNS for broadcasting and discovering sources,
  - DNS-SD content description and endpoint location
  - SPARQL 1.1 SDs and VoID descriptions of source content.
- Advantages:
  - approach addresses the NNEC needs
  - it is independent of a central registry,
  - eliminates the issue of network fragmentation.

Table of contents

Shortcomings

- Problem:

Shortcomings

- Problem:
  - `CONSTRUCT` queries are too unconstrained

Shortcomings

- Problem:
  - `CONSTRUCT` queries are too unconstrained
  - problem especially acute wrt. common predicates

Shortcomings

- Problem:
  - `CONSTRUCT` queries are too unconstrained
  - problem especially acute wrt. common predicates
  - consider e.g. `?a rdf:type ?o`

Shortcomings

- Problem:
  - `CONSTRUCT` queries are too unconstrained
  - problem especially acute wrt. common predicates
  - consider e.g. `?a rdf:type ?o`
  - this pattern will most likely be routed to *every* endpoint

Shortcomings

- Problem:
  - `CONSTRUCT` queries are too unconstrained
  - problem especially acute wrt. common predicates
  - consider e.g. `?a rdf:type ?o`
  - this pattern will most likely be routed to *every* endpoint
  - thus, downloading potentially huge amounts of data

Shortcomings

- Problem:
  - `CONSTRUCT` queries are too unconstrained
  - problem especially acute wrt. common predicates
  - consider e.g. `?a rdf:type ?o`
  - this pattern will most likely be routed to *every* endpoint
  - thus, downloading potentially huge amounts of data

Shortcomings

- Problem:
  - `CONSTRUCT` queries are too unconstrained
  - problem especially acute wrt. common predicates
  - consider e.g. `?a rdf:type ?o`
  - this pattern will most likely be routed to *every* endpoint
  - thus, downloading potentially huge amounts of data
- Proposed solution:

Shortcomings

- Problem:
  - `CONSTRUCT` queries are too unconstrained
  - problem especially acute wrt. common predicates
  - consider e.g. `?a rdf:type ?o`
  - this pattern will most likely be routed to *every* endpoint
  - thus, downloading potentially huge amounts of data
- Proposed solution:
  - assess the selectivity of triple patterns

Shortcomings

- Problem:
  - `CONSTRUCT` queries are too unconstrained
  - problem especially acute wrt. common predicates
  - consider e.g. `?a rdf:type ?o`
  - this pattern will most likely be routed to *every* endpoint
  - thus, downloading potentially huge amounts of data
- Proposed solution:
  - assess the selectivity of triple patterns
  - build the cropping incrementally

Shortcomings

- Problem:
  - `CONSTRUCT` queries are too unconstrained
  - problem especially acute wrt. common predicates
  - consider e.g. `?a rdf:type ?o`
  - this pattern will most likely be routed to *every* endpoint
  - thus, downloading potentially huge amounts of data
- Proposed solution:
  - assess the selectivity of triple patterns
  - build the cropping incrementally
  - within the confines of HTTP minimality

The selectivity of triple patterns

- Heuristics based on structural variations in triple patterns

The selectivity of triple patterns

- Heuristics based on structural variations in triple patterns
- $(s, p, o) \prec (s, ?, o) \prec (?, p, o) \prec \ldots \prec (?, ?, ?)$

The selectivity of triple patterns

- Heuristics based on structural variations in triple patterns
- $(s, p, o) \prec (s, ?, o) \prec (?, p, o) \prec \ldots \prec (?, ?, ?)$
- $\prec$-less is less likely to produce large results

The selectivity of triple patterns

- Heuristics based on structural variations in triple patterns
- $(s, p, o) \prec (s, ?, o) \prec (?, p, o) \prec \ldots \prec (?, ?, ?)$
- $\prec$-less is less likely to produce large results
- compare query patterns with the all-some lifting of $\prec$

The selectivity of triple patterns

- Heuristics based on structural variations in triple patterns
- $(s, p, o) \prec (s, ?, o) \prec (?, p, o) \prec \ldots \prec (?, ?, ?)$
- $\prec$-less is less likely to produce large results
- compare query patterns with the all-some lifting of $\prec$
- nonexclusive groups more weight than exclusive groups

The selectivity of triple patterns

- Heuristics based on structural variations in triple patterns
- $(s, p, o) \prec (s, ?, o) \prec (?, p, o) \prec \ldots \prec (?, ?, ?)$
- $\prec$-less is less likely to produce large results
- compare query patterns with the all-some lifting of $\prec$
- nonexclusive groups more weight than exclusive groups
- some patterns are preselected for the lowest level, e.g.

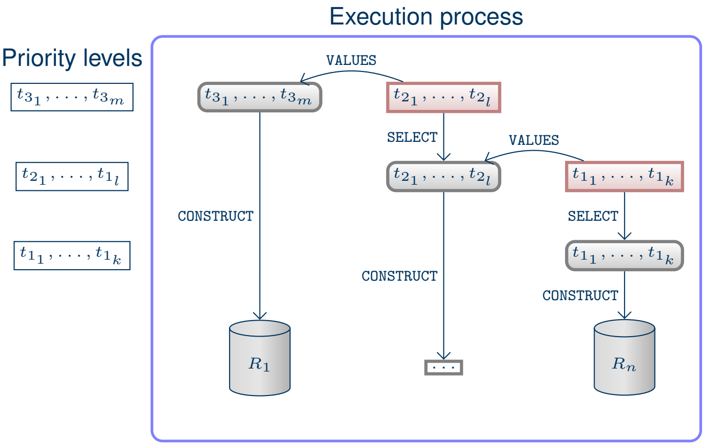
The selectivity of triple patterns

- Heuristics based on structural variations in triple patterns
- $(s, p, o) \prec (s, ?, o) \prec (?, p, o) \prec \ldots \prec (?, ?, ?)$
- $\prec$-less is less likely to produce large results
- compare query patterns with the all-some lifting of $\prec$
- nonexclusive groups more weight than exclusive groups
- some patterns are preselected for the lowest level, e.g.
  - `?s ?p ?o`

The selectivity of triple patterns

- Heuristics based on structural variations in triple patterns
- $(s, p, o) \prec (s, ?, o) \prec (?, p, o) \prec \ldots \prec (?, ?, ?)$
- $\prec$-less is less likely to produce large results
- compare query patterns with the all-some lifting of $\prec$
- nonexclusive groups more weight than exclusive groups
- some patterns are preselected for the lowest level, e.g.
  - `?s ?p ?o`
  - `?s rdf:type ?o`

Incremental construction of the cropping:



Summary of key properties

- federated query answering is provably sound/complete
- only one HTTP request is sent to each source
- query answering is tractable, i.e. takes only polynomially many computation steps
- source selection is dynamic and once-per-query

Current and future work

- Status:

Current and future work

- Status:
  - a formal theory of federation + join-order optimization

Current and future work

- Status:
  - a formal theory of federation + join-order optimization
  - a working implementation in Scala/Clojure

Current and future work

- Status:
  - a formal theory of federation + join-order optimization
  - a working implementation in Scala/Clojure
  - a demo that runs against real military databases

Current and future work

- Status:
  - a formal theory of federation + join-order optimization
  - a working implementation in Scala/Clojure
  - a demo that runs against real military databases
- Prioritized future work:

Current and future work

- Status:
  - a formal theory of federation + join-order optimization
  - a working implementation in Scala/Clojure
  - a demo that runs against real military databases

- Prioritized future work:
  - adapt the theory and the implementation to data streams

Current and future work

- Status:
  - a formal theory of federation + join-order optimization
  - a working implementation in Scala/Clojure
  - a demo that runs against real military databases
- Prioritized future work:
  - adapt the theory and the implementation to data streams
  - address challenges related to lifting of sensor data

Current and future work

- Status:
  - a formal theory of federation + join-order optimization
  - a working implementation in Scala/Clojure
  - a demo that runs against real military databases
- Prioritized future work:
  - adapt the theory and the implementation to data streams
  - address challenges related to lifting of sensor data
  - design a general purpose system around the current core