Supporting Evacuation Missions with Ontology-Based SPARQL Federation

Audun Stolpe, Jonas Halvorsen and Bjørn Jervell Hansen
Norwegian Defence Research Establishment (FFI)
P O Box 25
2027 Kjeller, Norway
Email: {audun.stolpe | jonas.halvorsen | bjorn-jervell.hansen}@ffi.no

***Abstract*—We study ontology-based SPARQL federation in support of coordinated action by deployed units in military operations. It is presumed that bandwidth is limited and unstable. Thus, we need an approach that generates few HTTP requests. Existing techniques employ join-order heuristics that may cause requests to multiply as a factor of the number of joins in a query. This can easily lead to an amount of traffic that exceeds network capacity. We propose an approach that builds an in-memory excerpt of the remote sources, sending one request to each source. A query is answered against this excerpt, which is a provably sound and complete representation of the sources wrt. query answering. The paper ends with a case study involving three military sources used for planning evacuation missions.**

I. Introduction

The planning of evacuation missions is a complex and important process in military operations. One of the most challenging aspects, is making all necessary information available to the decision makers. These information fragments will typically be distributed across different systems.

This is particularly the case when the military force conducts its operations according to network-based concepts, like NATO's Network Enabled Capability, henceforth NNEC [1]. The primary objective when conducting operations according to this concept, is to support the creation of a high degree of shared situational awareness among decision makers in order to obtain increased mission effectiveness. A prerequisite for achieving this, is extensive information sharing and a robust scheme for information integration, enabling decision makers to retrieve and utilize all relevant information when needed.

So far, the emphasis of the technical work on NNEC has been on how to make information available throughout the environment. However, in order for NNEC to be of use to decision makers, the challenge of establishing a robust scheme for information integration ultimately also needs to be addressed. This is the focus of the research reported on in this paper.

We present an information integration approach that combines query rewriting with data federation, and we study it in relation to an example from military evacuation planning based on live reporting of incidents over IP radio networks.

The main contribution of this paper consists in defining a novel federation strategy specifically designed for domains that share the general characteristics of this case. The most important characteristics are firstly that bandwidth is limited so the total communication costs induced by the number of HTTP request is a non-negligible factor, and secondly that the network topology is dynamic, i.e. sources may come and go. Our aim is thus to define a federation strategy that is sound and complete with respect to query answering, issues a minimal number of HTTP requests, and is compatible with run-time detection of sources.

The paper is organized as follow: In Section II we identify a list of tentative desiderata that our federation strategy should satisfy. The desiderata points to using an ontology-based data access paradigm, which is explained in Section III. Section IV outlines our solution, which is based on querying against an excerpt, or *cropping*, of the remote sources relative to an incoming query. The case study is presented in Section VI, and the main experiences drawn from the case study is presented in Section VII.

The paper assumes familiarity with W3C's Semantic Web technology stack, in particular RDF, OWL, and SPARQL. Readers not familiar with these technologies are referred to [2], [3], and [4] for an introduction.

II. Characteristics of the domain

The NNEC concept presupposes a network-based environment in which information about own and enemy units is typically distributed across several autonomous data sources contributed by coalition members. In order to support evacuation planning, these information fragments need to be integrated in order for the decision makers to obtain the highest possible degree of situational awareness. This involves tackling some idiosyncratic challenges:

- The information systems are in general semantically heterogeneous, especially in coalition operations, and cannot be accessed in a coherent and unified way,
- the underlying communication network often relies on IP radios, and is hampered by limited bandwith, latency, and limited range,

- the network topology is highly dynamic, meaning that information systems can appear and disappear at any time, and
- the shared information is mission-critical, which makes it crucial that the integration scheme yields correct and exhaustive data.

These characteristics means that we want to define an information integration approach that:

A allows a user to access available sources in a unified way,
B utilizes the available bandwith efficiently, particularly by restricting the number of HTTP requests to the remote sources,
C allows the relevant sources to be discovered at run-time, and
D guarantees the soundness and completeness of query answering.

III. Ontology-based data access

Based on the desiderata from section II, we decided to use an ontology to mitigate heterogeneities and to provide uniform access to the data. That is, we based our approach on the paradigm usually called *ontology-based data access* in which a conceptual model—the ontology—is used to express the relationship between the content of the respective sources, and to act as a single query interface towards them.

According to the W3C Web Ontology Working Group[1], an ontology defines a set of concepts or term used to describe and represent some domain of information in an abstract way that gives a formal semantics to the data in question. More specifically, an ontology gives the semantics of the data in the form of a set of logical axioms that explicate the relationship between classes of data items, and it enables computers to reason over the data as part of the process of answering a query. One particular form that this process can take, is that in which a query formulated in terms of the concepts of the ontology is successively refined until the query can be executed directly against the data. This is usually referred to as *query rewriting* and forms the basis for our approach, as explained in the next section.

Ontology-based data access is useful in all scenarios in which accessing data in a unified and coherent way is difficult. This may happen for several reasons. The data sources may have been developed for different purposes by different agencies or institutions, may not have a coherent design, and may not record similar types of information in the same manner. A well-designed ontology gives a unified view of the domain in terms of the *concepts* that are of interest to the user.

IV. Outline of approach

Our federation engine is designed to be suitable for a dynamic network topology, in accordance with our listed desiderata. To that end, sources are selected at query time based on the outcome of the reasoning process, by inspecting DNS records that are multicasted in the network (cf. section VI.). The entire federation process can thus be seen as comprised of two distinct steps. First, the query is rewritten into a query expressed directly in terms of the data according to the domain model expressed by the ontology. Next, the rewritten query is decomposed into sub-queries that yield a set of mutually exhaustive partial answers extracted from each of the selected sources.

It is not automatically the case, however, that the above mentioned steps are separable. That is, depending on the expressive power of the ontology language, reasoning may require data-access. This is not the case for the class of ontology languages that are *first-order rewritable* (cf. [5], [6]). This notion was first introduced by Calvanese et al. ([5]) in the context of the class of ontology languages called description logics. A description logic L—more generally an ontology language—is first-order rewritable if, for every ontology Σ expressed in L and a query Q , Q can be compiled into a first-order query Q_Σ that A) compiles *away* all concepts from the ontology Σ, and B) is such that given a data repository R, Q_Σ evaluated over R yields exactly the same result as Q evaluated against R and Σ.

First-order rewritable ontology languages is a crucial presupposition behind our approach. It is important precisely because it ensures that the reasoning process can be decoupled from data access. This has two hugely beneficial consequences: First, the complexity of reasoning remains unaffected as data size increases. In other words the computation time allocated to reasoning will not vary with changes in the network topology and/or availability of sources—recall that we do not assume these to stay fixed. Secondly, since reasoning can be decoupled from data access, it does not affect the federation process *per se*. That is, source selection can be performed independently of the reasoning process, which means, among other things, that a query which is run repeatedly will only have to be rewritten once.

As always there is a price to pay, though. The property of first-order rewritability imposes a serious constraint on the expressivity of an ontology language, and, as explained in section VII, must be very carefully selected in order to be able capture the salient aspects of our case. In particular, it turns out that none of the standard fragments of the W3C-endorsed ontology language OWL will do.

As regards HTTP-minimality (by which we mean keeping the number of required HTTP requests as low as possible), we found existing approaches to federated query processing not to be well suited. One of the main reasons is that they all rely on forms of join-order heuristics that tends to multiply the number of HTTP requests as a factor of the depth and number of joins: a standard distributed join algorithm will evaluate a query iteratively one triple at a time, while propagating values in a nested loop join fashion. This multiplies HTTP request in proportion to the result sets returned by evaluating each join

[1] http://www.w3.org/2001/sw/WebOnt/

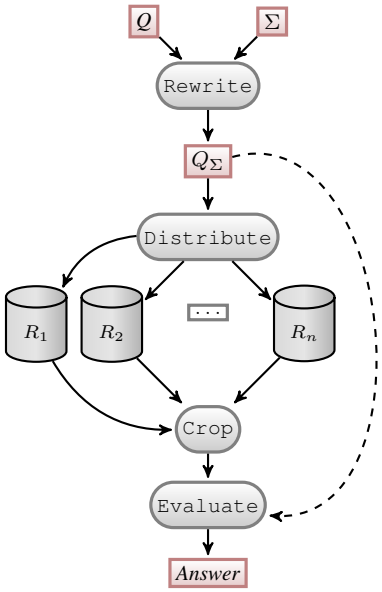

Figure 1. System overview

argument.[2] Admittedly, there are several improved versions of this algorithm on offer. The bound join technique implemented in FedX [9], for instance, groups several instances of a join argument in a single subquery using the SPARQL `UNION` construct. This reduces the number of request with a factor equivalent to the the number of instances in the grouped query (ibid.).[3] Yet, experimental evaluation shows that the number of HTTP request can still grow quite fast in the number of joins.[4][5]. What is common to all these approaches is that the number of HTTP request varies in the number of results returned by the sub-queries. It is a design goal of our approach, in contrast, to make a factor of the size of the query only.

To that end, we designed our federation engine to evaluate the query, not against the sources directly, but against an excerpt, or *cropping* as we call it, that is pulled from the sources by sending a single HTTP-request to each. Unlike traditional warehousing strategies, however, our local copy is not persisted, but exists only in-memory for the duration of the query execution process. It is essentially a snapshot of that part of the remote sources which is relevant for answering the query in question. In realistic cases, the cropping is much smaller than the total amount of data that it is extracted from (see Section V-A).

An overview of the resulting system, is shown in Figure 1: the system takes as input a SPARQL query Q, and a collection of aligned ontologies Σ, which are used by the rewriter to produce the query Q_Σ. This rewritten query is next handed to the federator component which performs service discovery at run-time (cf. Section VI) to identify live and relevant sources. Relevance here means signature overlap, where a signature is understood as a set of RDF properties. The extent of the overlap between the signature of the query and the signature of a given endpoint determines a SPARQL `CONSTRUCT` query which will be routed to that endpoint.

The `CONSTRUCT` queries are designed to adhere to a logical form which is sufficiently structured to enable us to guarantee the soundness and completeness of the query answering process wrt. the set of sources $\mathscr{R}$, as explained in more detail in the next sections. Taken together with the obvious minimality of our approach wrt. the number of HTTP requests—only a single request is sent to each source—as well as the relevance-based per-query discovery of sources, we conclude that our approach meets all our tentative desiderata A) to D).

V. Properties of the Cropping

In this section we formally define the notion of the cropping of a distributed set of sources $\mathscr{R}$ relative to a query Q, and we state its essential properties. We shall assume familiarity with SPARQL syntax and semantics (cf. [12]).

Notation. We use R_i, where i is in some index set I, to denote RDF graphs—variably referred to as sources, repositories or endpoints. A SPARQL `SELECT` query is a pair $\langle P, \vec{x}\rangle$, where P is a SPARQL graph pattern and $\vec{x}$ a vector of elements of variables. Similarly, a `CONSTRUCT` query is a tuple $\langle T, P\rangle$, where T is a basic graph pattern and P is a union of such. T will be identified with the `CONSTRUCT` block of the query, aka. the *template*, and P with the `WHERE` block, aka. the *query pattern*. We shall allow ourselves the convenience of blurring the distinction between SPARQL queries on the one hand and sets and families of triple patterns on the other. Where $Q := \langle P, \vec{x}\rangle$ is a `SELECT` query and G an RDF graph we denote the result of evaluating Q against G as $Q(G)$, and similarly for `CONSTRUCT` queries. The proofs of the claims that follow can be found in technical report [13].

As mentioned in the previous section, our approach to federation is signature-based in the sense that the RDF properties that are found in a query are used for routing different sub-queries to different endpoints. This is a common strategy (cf. [8], [9]) for which we claim no originality. Now, given a query pattern P the *relevant subset* of P in relation to a source R_i is defined as the maximal subset of P whose signature is contained in the signature of R_i. We shall denote this set as $\rho(P, i)$.

Recapitulating briefly, our federation engine is designed to be HTTP-minimal, as well as sound and complete wrt. to query answering over the selected sources. A strategy that supports all three is to execute the query against an *in-memory representation* of the remote sources rather than against the sources themselves. More specifically our federation engine routes a single `CONSTRUCT` query to each of the selected sources—achieving HTTP minimality—whereas the logical form of this construct query is defined in such a manner as

[2] DARQ [7] and SPLENDID [8] both implement a version of this algorithm.
[3] Similar techniques are considered in [8] and [10]
[4] See e.g. the results for FedX on Life Sciences 3 query from the FedBench suite.
[5] Another notable optimization is the star-shaped pattern technique of [11]. Numbers of requests are however not reported in this study.

to guarantee that the answer to the query assembled from the selected sources is both correct and complete with respect to those sources. Here soundness and completeness means that if $\mathscr{R}$ is a set of sources selected for federation, then the answer that the federator provides to a query Q should be exactly the same as the one that would be obtained were Q to be evaluated conventionally over a single repository holding the union of the data sets in $\mathscr{R}$. To the best of our knowledge, our strategy is currently the only one that guarantees that this is the case.

The logical form in question is in turn defined by distinguishing between *exclusive* and *non-exclusive* triples in a query pattern P. Exclusive triples are those that are satisfied, if at all, at one endpoint only. Non-exclusive triples, on the other hand, may be satisfied by two or more. Exclusive triples can safely be grouped together and executed against the source for which it is exclusive in as a single conjunctive pattern. Non-exclusive triples, however, must be shipped to the remote sources as separate `UNION` clauses. This holds even if a group of triple patterns are relevant to exactly the same sources since an answer to the original query may require joining triples across these sources. This gives rise to the following definition of *the set of clauses induced by P and R_i*:

Definition 1 (Clause set): For R_i a source and P a query pattern: $s(P,i) := \{\epsilon(P,i)\} \cup \{\{t\} : t \in \rho(P,i) \setminus \epsilon(P,i)\}$

Here $\epsilon(P,i)$ denotes the exclusive group of a pattern P relative to R_i.

Now, the basic idea behind our federation strategy is to use the set of clauses induced by P and R_i to define a `CONSTRUCT` query that extrapolates the part of R_i that is relevant for answering P. The most straightforward way to do that may seem to be to use the clause set itself as a query pattern, whilst using the set-theoretic *union* of its elements as a template. Call this the *naive* strategy. Interestingly, the naive strategy, whilst complete, is *not* sound. Consider the following rather abstract example:

Example 1: Let G be the RDF graph containing only the two triples $\mathtt{s} := (\mathtt{c1}, \mathtt{p}, \mathtt{d1})$ and $\mathtt{t} := (\mathtt{c2}, \mathtt{q}, \mathtt{d2})$, and assume a clause-set $\{\{(?\mathtt{s}, \mathtt{p}, ?\mathtt{o})\}, \{(?\mathtt{s}, \mathtt{q}, ?\mathtt{s})\}\}$. The corresponding naive `CONSTRUCT` query is:

```
CONSTRUCT {?s q ?o. ?s p ?o.}
WHERE {{?s p ?o} UNION {?s q ?o}}
```

Executing this query against G will produce a graph containing the triple $(\mathtt{c1}, \mathtt{q}, \mathtt{d1})$.

The example shows that the naive strategy may create bindings in the resulting graph that do not exist in the graph that is queried. To counteract this effect it is necessary to standardize apart the elements of the clause-sets before using taking the union and using it as a `CONSTRUCT` template. To this end we introduce the notion of a *separation function*:

Definition 2 (Separation function): Let $S := \{c_1, \ldots, c_n\}$ be a clause set, and let σ_i be a uniform substitution of variables for variables in c_i. A separation function f for S is a function s. t. 1) $f(S) = \{\sigma_1(c_1), \ldots, \sigma_n(c_n)\}$, and 2) $\sigma_j(?x) \neq \sigma_k(?x)$ for every $?x \in dom(\sigma_j) \cap dom(\sigma_k)$.

Our `CONSTRUCT` queries now become:

Definition 3: For a set of sources $\mathscr{R} := \{R_i\}_{i \in I}$ and a query pattern P: $\mathscr{C}(P,i) = \langle \bigcup f(s(P,i)), f(s(P,i)) \rangle$ where f is some separation function for $s(P,i)$.

Example 2: Suppose we have two endpoints `JOCWatch` and `MedWatch`,[6] and the following conjunctive graph pattern P:

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

Suppose further that each property prefixed by `medics` belongs to the signature of `MedWatch`, that each property prefixed by `jocw` belongs to the signature of `JOCWatch`, and that the `jocw:status` property belongs to both. The queries that are routed to the respective endpoints are then:

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.}}
```

JOCWatch:

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

The cropping may now be defined as follows:

Definition 4 (Cropping): Put $Q := \langle P, \vec{x} \rangle$ and $\mathscr{R} = \{R_i\}_{i \in I}$. Then $\mathscr{A}^{\mathscr{R}}_Q := \bigcup_{i \in I} \mathscr{C}(P,i)(R_i)$.

We now have:

Theorem 1 (Soundness/Completeness): Let $\mathscr{R}$ be any set of sources, then $Q(\bigcup \mathscr{R}) = Q(\mathscr{A}^{\mathscr{R}}_Q)$ for any `SELECT` query Q.

Note that here Q is the `SELECT` query that is being posed to the system, whereas $\mathscr{A}^{\mathscr{R}}_Q$, i.e. the cropping, is the result of assembling the results of the `CONSTRUCT` queries that are required for providing an excerpt guaranteed to answer it.

As regards time complexity, since every `CONSTRUCT` query $\mathscr{C}(P,i)$ is in union normal form, Corollary 1 of [12] immediately entails that the cropping can be built efficiently. In our actual implementation, the `CONSTRUCT` queries that are allocated to the respective endpoints are, moreover, all executed in parallel, so time is not a precarious measure.

A. Restricting the size of the cropping

Although time is not a precarious measure, the size of result sets quickly becomes an issue. The `CONSTRUCT` queries that are passed around to the remote endpoints, if not constrained,

[6] These systems are described in VI

may well distribute a triple pattern $(?\texttt{a}, \texttt{rdf:type}, ?\texttt{b})$ to all remote endpoints, in effect requesting huge chunks of the data contained in each.

Now, there is no need in our approach for join-ordering heuristics in the conventional sense, since, per the approach, joins are either executed remotely, or executed locally by a standard query processing engine after the cropping has been built. Rather, what we do, is to build the cropping incrementally by assessing the relative selectivity of triple patterns and processing the most selective ones first. We can only describe this procedure in general outline here:

The selectivity of a triple pattern may be assessed along several dimensions. For instance, studies show that a triple pattern with a literal in object position will usually be more selective than one with a URL in the same position [14]. Moreover, triple patterns can be ordered in a plausible sequence of decreasing selectivity based on the distribution, and position, of variables in the pattern (? denotes a variable):

$(s,p,o) \prec (s,?,o) \prec (?,p,o) \prec (s,p,?) \prec (?,?,o) \prec (s,?,?) \prec (?,p,?) \prec (?,?,?)$

The entire set of heuristic rules that we have used in our solution can be found in [14].

Now, the idea is to build the cropping in layers by employing the following three-step procedure: 1) construct the graph corresponding to the most selective patterns pertaining to each endpoint 2) extract variable bindings from the cropping so far, and 3) pass them on to the next iteration as constraints for the next round of queries.

Step 2, the extraction of variable bindings, is realized by re-using the triple patterns as `SELECT` queries that are evaluated against the cropping as it exists so far, whereas the propagation of values from one layer to the next is realized with the `VALUES` feature of SPARQL 1.1, which allows a set of bindings to be shipped with a query in order to constrain the answers.

Our procedure is designed to treat each exclusive group as an atomic unit, since exclusive groups are likely to be more selective as a *set*. For the same reason, they are given maximum priority, That is, the first layer of the incremental construction of the cropping consists of the result of executing the exclusive groups as `CONSTRUCT` queries. The subsequent layers are then constructed from the non-exclusive patterns by rating them according to the heuristic criteria.

This procedure is sufficient to ensure that very unconstrained patterns such as $(?\texttt{s}, ?\texttt{p}, ?\texttt{o})$ will be processed late, when bindings are available for some of its variables. It can also be tuned to give low priority to predicates from existing RDF vocabularies that are known to have a low selectivity rate, such as e.g. $\texttt{rdf:type}$ or $\texttt{dcterms:title}$. The procedure preserves the soundness and completeness of the cropping wrt. the underlying sources, and, although the number of HTTP request is no longer minimal it is constant in a small factor of the number of triple patterns in the original query.

VI. Case

The case we use to exemplify our approach is based upon the following scenario: A military analyst is monitoring planned medical evacuation flight missions, and is on the lookout for missions that might be threatened by enemy activity. If this is the case, she is also interested in finding friendly units able to counter the particular threat. More specifically, the information requirement is the following: *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.*

Normally, in order to obtain an answer to this information requirement, the analyst has to keep an eye on several systems, as information about evacuation flights and information about enemy activity are usually not kept in the same system. With the aid of a system as outlined so far in this paper, however, the analyst can pose a query formulating what she is looking for and let the information integration system take care of the rest.

To evaluate the approach and the case outlined above, we conducted an experiment at NATO CWIX 2013[7] set as close as possible to the dynamic and multinational environment of NNEC. The experiment involved three operational information systems: 1) `JOCWatch`, information on incidents of relevance to the command in an event log, 2) `MedWatch`, a system for medical mission tracking designed to support the planning, logging and monitoring of medical evacuation missions, and 3) `Track Source`, a unit tracking service providing times-tamped geopositional information regarding friendly units in the field. The information in `MedWatch` and `JOCWatch` were made available through SPARQL endpoints by using D2R [15], while the Track Source service had a native SPARQL interface. In addition, all sources were supplied with a service description according to the SPARQL 1.1 specification, and each source made available its ontology at an URL described in the service description.

Our prototype system performed service discovery using

- mDNS[8] for broadcasting and discovering the presence of information sources,
- DNS-SD[9] for high-level description of the source in terms of pointers to query endpoint location and content description location, and
- the SPARQL 1.1 service description and VoID[10] vocabularies for describing source content.

This approach addressed the NNEC needs as outlined in section II, and had the advantage that it was independent of a central registry, thus eliminating the issue of network

[7] Coalition Warrior Interoperability eXploration, eXperimentation and eXamination, eXercise – an annual NATO event aimed at improving alliance-wide interoperability

[8] http://www.multicastdns.org/

[9] http://www.dns-sd.org/

[10] http://www.w3.org/TR/void/

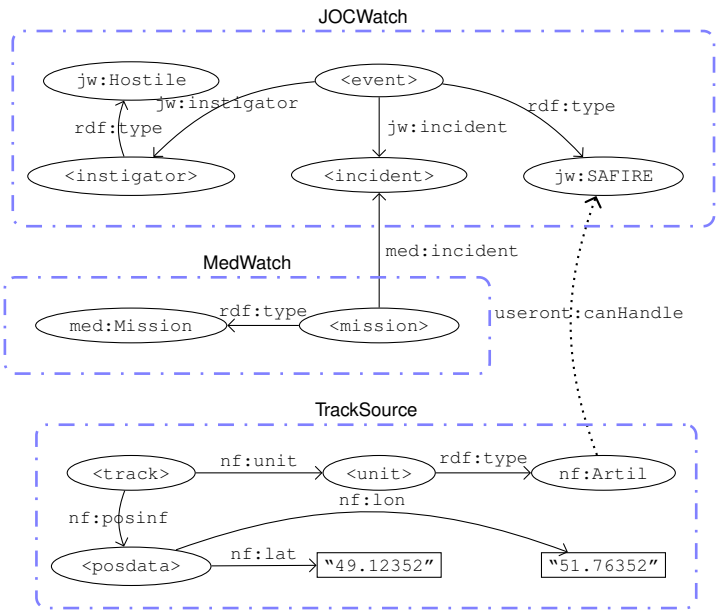

Figure 2. Conceptual relationship between data sources

Concept	Definition
ThreatenedMission	MedWatch missions that are related to a ThreateningIncident
ThreateningIncident	All JOCWatch incidents that are related to a ThreateningEvent
ThreateningEvent	All JOCWatch events that are both a MilitaryOperation (from the JOCWatch ontology) and a HostileEvent
HostileEvent	All events that has a HostileInstigator
HostileInstigator	All event participants that are classified as being hostile.
Relation	**Definition**
canHandle	A relation between a military unit type and the type of events those units types are equipped to handle.
hasEvent	If a mission involves an incident, and there exists an event that belongs to the same incident (inverse property), then the event is also related to the mission

Table I. RELEVANT DEFINITIONS IN THE USER ONTOLOGY

```
SELECT ?mission ?unit
WHERE{
   ?mission a useront:ThreatenedMission.
   ?mission useront:hasEvent ?event.
   ?event a useront:ThreateningEvent.
   ?event wgs84:lat ?elat.
   ?event wgs84:long ?elong.
   ?unit useront:canHandle ?event.
   ?unit useront:hasPosition ?pos.
   ?pos wgs84:lat ?ulat.
   ?pos wgs84:long ?ulong.
}
```

Figure 3. SPARQL query representing the information request

fragmentation.

The relationship between the information sources in our experiment is illustrated in Figure 2. In the figure we see that MedWatch missions are (potentially) related to JOCWatch events through a shared incident. Furthermore, the JOCWatch events are typed according to category e.g. as a SAFIRE event, which is an event that involves hostile surface-to-air fire. In the figure we also see that units in the Track Source are typed (e.g. as Artillery), and that it contains positional data. Additionally, in the figure, a stipulated line is drawn from Artillery to SAFIRE, indicating that, units of the former kind are equipped to counter those of the latter. This relation does not actually belong to the data, but is defined in the user ontology useront, and is key to the formulation of the user's information need.

The experiment included four main ontologies: a JOCWatch ontology, a MedWatch ontology, a Track Source ontology, and an ontology containing the concepts used by the user of the system.

In this particular case, the user ontology is derived from and expresses the data models of the respective sources. We are thus assuming that, although available sources may come and go, we know their data models. This is not overly unrealistic, since NATO-wide standardization is part of the NNEC concept. The assumption means that we do not have to match ontologies at run-time. A user ontology less tightly coupled with the source ontologies and run-time ontology matching is something we plan to look more into in future work.

Given the user ontology, the information requirement described earlier can now be expressed by the query in Figure 3. [11]

Here ThreateningEvent, hasEvent, canHandle, and hasPosition are terms specific to the user's vocabulary, see table VI. Posing this query to any of the information sources would not return any answers. In our experiment, this query was decomposed and distributed as per the approach outlined earlier.

The main motivation behind this experiment was to test whether it is feasible to provide a decision maker with the means to request information using her own terms and without presupposing detailed knowledge about a fixed set of sources. This creates a coupling between the requesting system and the information sources that is loose enough to adapt to a changing network topology, something that should be highly relevant in the NNEC environment. Our strategy of combining once-per-query federation with rewriting worked well for our sample case, and proves that the idea is sound in general outline. To be sure, if the system is to to scale well—both in terms of efficiency and usability—there are some serious issues that need to be addressed having to do with the expressiveness of the ontology language and the complexity of reasoning in it. We record some findings in the next section.

[11] In reality, we apply filtering of friendly units based on distance from events using the haversine formula and a threshold value. As this does not contribute to understanding the general approach we have left it out of the example.

VII. Experiences and Observations

As explained in section IV, it is an essential presupposition of our approach to federation that the ontology that is used to provide access to the underlying sources be expressed in a first-order rewritable language. This is necessary in order to separate reasoning from source selection, thus making the system able to adapt to a dynamic network topology by selecting sources at run-time.

Yet, it is not given that the structure and relationship between the sources that we selected for our case-study, as illustrated in Figure 2, can in fact be expressed in a first-order rewritable language.

Choosing an ontology language in the DL-Lite family of description logics would have been natural for several reasons: First,these languages are specifically designed to stay within the boundaries of first-order rewritability. Secondly, DL-Lite forms the basis of the W3C-endorsed QL language profile of OWL 2, and so has an XML serialization, and enjoys the status of an official recommendation. Finally, several efficient rewriters already exist for the DL-Lite family of languages, which, if we could use them, would of course leverage the burden of implementing our own federation engine.

As it turns out, however, our case cannot be expressed in any of the standard OWL2 profiles, nor in any other description logic we are currently aware of. This is due to a combination of features exemplified by the structure of our sources. Referring to Figure 2 there are mainly two sources of expressive complexity:

1) As indicated by the stipulated arrow labelled `useront:canHandle` connecting `TrackSource` to the `JOCWatch` database, we wish to add axioms to the ontology that classify which kind of vehicle or unit that is equipped to counter which kind of hostile event—for instance artillery in the case of a surface-to-air attack. Encoding this knowledge in the ontology is necessary in order to enable the user to query the sources for available military support within a given diameter from a threatened position. However, it requires that we be able to state in the ontology that certain combinations of unit types and events constitute sufficient conditions for the unit and event in question to stand in the `canHandle` relation. Stated more formally, we need to have axioms of the form $\forall x \forall y. Artillery(x) \wedge SAFIRE(y) \rightarrow canHandle(x, y)$, for all the appropriate combinations of units and events.

2) Presupposing that the `canHandle` relation has been axiomatized, we further need to express in the ontology that finding the position of a unit involves traversing the `TrackSource` graph from the reported latitudes and longitudes through the relations `nf:posinf`, `nf:unit` and `rdf:type` via `useront:canHandle` to an associated hostile event. This is a fairly long and intricate path that requires traversing relations forwards as well as backwards (i.e. traversing the inverse of the relation).

The problem with 1) is that it has a binary predicate in the conclusion. For that reason, it cannot be expressed as a class inclusion axiom. A description logic axiom is either a class axiom or a relationship axiom (aka. role axiom) but cannot be a mix of the two. Indeed, a class inclusion axiom—irrespective of the particular brand of description logic that is being used—cannot, by design, express cross-references between antecedent and consequent in two or more variables, as our axioms require.

Description logics typically allow us to state axioms like the following (in description logic notation):

- $\texttt{Artillery} \sqsubseteq \exists \texttt{canHandle.SAFFIRE}$
- $\texttt{Artillery} \sqsubseteq \forall \texttt{canHandle.SAFFIRE}$

At first glance, these may seem to come close to what we wish to say, but that is not really the case. The first says that an artillery unit can handle *some* surface-to-air-fire event, but it does not identify the event. The second says that an artillery unit can *only* handle surface-to-air-fire events, although it may not be able to handle *all* of them. What we wish to say though us that *all* artillery units can handle *all* surface-to-air-fire events.

Taking stock, there is thus, to the best of our knowledge, no first-order rewritable description logic capable of expressing the structure and interrelationship between our selected sample of military information sources. As it turns out, though, there is a different family of ontology languages altogether that *is* sufficiently expressive for our needs, namely the family of *general existential rules* aka. *existential datalog* [6], more specifically the language of *weakly recursive datalog*. Weakly recursive datalog is strictly more expressive than any first-order rewritable description logic—and more importantly, it is sufficiently expressive to express 1) and 2) above, thus capturing the salient features of our case. Our federation engine is therefore equipped with a rewriter that expects an ontology to be encoded in weakly recursive datalog.

Although, this choice is more or less forced upon us by the characteristics of the case, it does not mean of course that the choice of recursive datalog as our ontology language does not come with its own set of drawbacks. First of all, existential datalog in general does not currently have the kind of institutionalized support that the OWL family of languages enjoys. Secondly, and much for the same reason, it has far less endorsement from the software industry in terms of tool support.

In fact, we could not find an existing rewriter for weakly recursive datalog, and therefore had to build one from scratch. Alas, implementing a correct rewriter does not entail that one has implemented an efficient one, and although queries over weakly recursive datalog ontologies are first-order rewritable, the size of the rewriting itself may be exponential in the size of the original query. Thus, without a considerable amount of research being devoted to optimization, the rewriter is not

likely to perform well for any large class of cases. Theoretical results are encouraging, though. In particular, results from [16] shows that there is a minimal rewriting of any query over a set of weakly recursive datalog rules. Computing such a minimum, however, remains a topic for future research.

VIII. Related Work

Several studies have addressed the problem of decomposing a SPARQL query into sub-queries that can be allocated to a distributed set of remote sources. Notable examples include [17], [7] [18], [19], [20], [9], [11] and [8]. All of these studies belong to what we would call the join-order heuristics paradigm, and, unlike the present paper, none gives particular attention to establishing framework that is both sound/complete and request-minimal. Moreover, the listed reports focus exclusively on federating queries that are expressed directly in terms of the data. To the best of our knowledge there are very few contributions that address the question of how to combine query federation with reasoning, where reasoning cuts across several sources.

IX. Conclusion

In this paper we have established a sound, complete and request-minimal baseline for query federation. Our approach is signature-based and compatible with a run-time selection of sources. It is therefore particularly suitable for domains that are characterized by low bandwidth and a dynamic network topology. We have further described an example from military evacuation planning to illustrate the usefulness of the approach. In order to mitigate the heterogeneities between sources, as well as to present the data in a vocabulary that is familiar to the user, we found it expedient to use an ontology to provide a unifying layer above the information sources. Any incoming query is therefore rewritten according to the ontology before being passed on to the sources. However, we have that ontology-based data access is not coherent with our federation strategy unless the ontology is formulated in a language that is first-order rewritable such that reasoning can be decoupled from data access. In realistic cases like ours, one quickly transcends the expressive capabilities of familiar first-order rewritable OWL fragments such as OWL2-QL. In our case we overcame this limitation by resorting to a decidable fragment of existential datalog.

Acknowledgments

The NATO systems participating in the experiments reported in this paper was made available to us by the NATO C3 Agency.

References

[1] P. Bartolomasi, T. Buckman, A. Campell, J. Grainger, J. Mahaffey, R. Marchand, O. Kruidhof, C. Shawcross, and K. Veum, "NATO Network Enabled Capability Feasibility Study, Version 2.0," NATO C3 Agency, Tech. Rep., October 2005.

[2] W3C, "RDF Primer," http://www.w3.org/TR/rdf-primer/, February 2004.

[3] ——, "OWL 2 Web Ontology Language Primer," http://www.w3.org/TR/2009/REC-owl2-primer-20091027/, October 2009.

[4] ——, "SPARQL 1.1 Query Language," http://www.w3.org/TR/sparql11-query/, March 2013.

[5] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati, "Ontologies and Databases: The DL-Lite Approach," in *Semantic Technologies for Informations Systems*. Springer, 2009.

[6] G. Gottlob, G. Orsi, and A. Pieris, "Ontological Queries: Rewriting and Optimization (Extended Version)," *Computing Research Repository*, vol. abs/1112.0343, 2011.

[7] B. Quilitz and U. Leser, "Querying distributed RDF data sources with SPARQL," in *Proceedings of the 5th European Semantic Web Conference (ESWC '08)*, 2008.

[8] O. Görlitz and S. Staab, "SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions," in *Proceedings of the 2nd International Workshop on Consuming Linked Data (COLD 2011)*, 2011.

[9] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt, "FedX: Optimization Techniques for Federated Query Processing on Linked Data," in *Proceedings of the 10th International Semantic Web Conference (ISWC 2011)*, 2011.

[10] J. Zemanek and S. Schenk, "Optimizing SPARQL Queries over Disparate RDF Data Sources through Distributed Semi-Joins," in *International Semantic Web Conference (Posters and Demos)*, ser. CEUR Workshop Proceedings, vol. 401, 2008.

[11] G. Montoya, M.-E. Vidal, and M. Acosta, "A heuristic-based approach for planning federated sparql queries." in *COLD*, ser. CEUR Workshop Proceedings, J. Sequeda, A. Harth, and O. Hartig, Eds., vol. 905. CEUR-WS.org, 2012.

[12] M. Arenas, C. Gutierrez, and J. Pérez, "Foundations of RDF Databases," in *Reasoning Web. Semantic Technologies for Information Systems*. Springer, 2009.

[13] B. J. Hansen, J. Halvorsen, and A. Stolpe, "Information integration experiment at NATO CWIX 2012," Norwegian Defence Research Establishment (FFI), FFI/RAPPORT-2012/01543, 2012.

[14] P. Tsialiamanis, L. Sidirourgos, I. Fundulaki, V. Christophides, and P. Boncz, "Heuristics-based query optimisation for sparql," ser. Proceedings of EDBT '12. ACM, 2012.

[15] C. Bizer and R. Cyganiak, "Publishing Relational Databases on the Semantic Web," http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/, August 2009.

[16] C. Civili and R. Rosati, "A broad class of first-order rewritable tuple-generating dependencies," in *Datalog in Academia and Industry*, ser. Lecture Notes in Computer Science, P. Barceló and R. Pichler, Eds. Springer Berlin Heidelberg, 2012, vol. 7494, pp. 68–80.

[17] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus, "Anapsid: An adaptive query processing engine for sparql endpoints." in *Proceedings of the 10th International Semantic Web Conference (ISWC 2011)*, 2011.

[18] C. Basca and A. Bernstein, "Avalanche: Putting the spirit of the web back into semantic web querying," in *ISWC Posters&Demos*, 2010.

[19] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, and J. Umbrich, "Data summaries for on-demand queries over linked data," in *Proceedings of the 19th international conference on World wide web*, ser. Proceedings WWW '10. New York, NY, USA: ACM, 2010, pp. 411–420.

[20] Y. Li and J. Heflin, "Using reformulation trees to optimize queries over distributed heterogeneous sources," in *Proceedings of 9th the International Semantic Web Conference (ISWC)*, ser. Proceedings ISWC'10. Springer-Verlag, 2010.