A Framework for Situation-Aware Access Control in Federated Data-as-a-Service Systems Based on Query Rewriting

Samson Oni*, Zhiyuan Chen*, Adina Crainiceanu†, Karuna P Joshi* and Don Needham†
* *University of Maryland Baltimore County, Baltimore, MD*
Email: {soni5, zhchen, karuna.joshi}@umbc.edu
† *US Naval Academy, Annapolis, MD*
Email: {adina, needham}@usna.edu

***Abstract*—Organizations often need to share mission-dependent data in a secure and flexible way. Examples include contact tracing for a contagious disease such as COVID-19, maritime search and rescue operations, or creating a collaborative bid for a contract. In such examples, the ability to access data may need to change dynamically, depending on the situation of a mission (e.g., whether a person tested positive for a disease, a ship is in distress, or a bid offer with given properties needs to be created). We present a novel framework to enable situation-aware access control in a federated Data-as-a-Service architecture by using semantic web technologies. Our framework allows distributed query rewriting and semantic reasoning that automatically adds situation based constraints to ensure that users can only see results that they are allowed to access. We have validated our framework by applying it to two dynamic use cases: maritime search and rescue operations and contact tracing for surveillance of a contagious disease. This paper details our implemented solution and experimental results of the two use cases. Our framework can be adopted by organizations that need to share sensitive data securely during dynamic, limited duration scenarios.**

***Keywords*-Ontology, Federated Systems, Data-as-a-Service, Semantic Web, Access Control**

I. Introduction

Complex, evolving situations often require close interaction between multiple entities. An example is maritime search and rescue (SAR) missions which often involve collaboration between military organizations, government agencies, private vessels, and even foreign vessels. Another example is contact tracing of contagious disease exposure which involves multiple entities like contact tracers, healthcare providers, and private businesses such as airlines and hotels that provide data (e.g. passenger lists) to help identify people in close contact with someone diagnosed with the disease.

In these limited duration, dynamic scenarios, each entity typically has data that needs to be kept private, as well as data that needs to be shared with other collaborators to accomplish the mission. Current approaches to data sharing are centered around situation-aware access control, also called policy-based or attribute-based access control [1], [2]. Entities may join the mission at any time, and data access decisions depend on situations like whether a ship is in distress or a person tests positive for a disease. However, these approaches only consider context-awareness within a single organization and often do not account for the challenges of securely sharing specific data elements between multiple organizations where each organization may have different data sharing policies.

One solution to supporting data sharing in collaborative scenarios is a federated Data-as-a-Service system. Multiple members form a federation and each provides a Data-as-a-Service interface to allow other members to query data stored at one or possibly multiple members' local data store. There are two main challenges to supporting situation-aware access control in a federated Data-as-a-Service system: 1) supporting distributed reasoning in evaluating access control rules, as this may require data from multiple members, and 2) efficiency, especially in cases such as maritime SAR with typically poor network connectivity. Existing work supporting distributed reasoning [3] fails to address the first challenge as highly specialized systems are required, making them hard to deploy within an access-control framework. Although significant work has been done on query rewriting [2], [4]–[10], most of such work either does not focus on access control or does not address the problem in a distributed environment.

We have developed a novel framework that allows policy based situation aware access in federated Data-as-a-Service systems. This framework facilitates automated query-rewriting by adding necessary constraints to the original query by checking organization's access control policies. We move away from a distributed reasoning focus into a distributed query focus. Since many federated Data-as-a-Service systems already support federated queries, our approach can be easily integrated into existing systems. Our approach also addresses the efficiency issue by using a peer-to-peer architecture such that fragments of a rewritten query can be executed locally, where data resides, reducing communication overhead.

We make the following contributions: 1) a semantic web based approach to defining situation-aware access control policies. We use maritime search and rescue and contact tracing for surveillance of a contagious disease as two

2474-2473/20/$31.00 ©2020 IEEE
DOI 10.1109/SCC49832.2020.00008

use cases, and develop an ontology and a set of rules to define access control policies for each use case; 2) a distributed reasoning framework based on query rewriting that enforces these rules and can be easily integrated with existing federated Data-as-a-Service architecture; 3) a peer-to-peer architecture that reduces communication overhead and is suitable for application scenarios, such as maritime SAR, with limited network bandwidth.

In our preliminary work [11], we proposed an initial proof-of-concept system for a situation-aware access control framework. In this paper, we substantially extend our preliminary work, including a new query rewriting algorithm, a new use case, and extensive experiments.

The rest of the paper is organized as follows. Section II describes the overall architecture of our framework. Section III describes the semantic approach to representing situation-aware access control rules, and the ontology and sample rules developed for two use cases. Section IV presents our query-rewriting based distributed reasoning framework. Section V describes implementation and experimental results. Section VI summarizes related work. Section VII gives our conclusions and discusses future work.

II. System Architecture

Background: In this paper we assume that each member in the system stores data in a common RDF format [12]. RDF data are represented as triples, each containing a subject, a predicate, and an object. For example, John (subject) has the role (predicate) of Captain of a ship (object). The structure of RDF data can be represented in an ontology language such as OWL [13]. The data can be queried using a SQL-like language such as SPARQL [14].

Consider the SPARQL query QS_1 below which returns all data from members of a SAR center.

```
QS1: PREFIX ns: <http://www.sar.org/ns#>
SELECT  ?Result
WHERE {
?Organization ns:isMemberOf ?SARCenter .
?Organization ns:has ?Result . }
```

SPARQL queries have a SELECT clause that indicates the variables returned by the query and a WHERE clause which specifies conditions the results need to satisfy.

In this paper we only consider Basic Graph Pattern (BGP) SPARQL queries in which the WHERE clause consists of conjunctions of triple patterns. Each triple pattern has a subject, predicate, and object, but any of these components can be a literal (e.g, ns:Location) or a variable (e.g. ?Result).

We consider a federated data-as-a-service system where each member provides a *service endpoint* that supports SPARQL 1.1 queries. A SPARQL 1.1 query can specify parts of the query to be executed at each SPARQL service endpoint and the results from all endpoints are put together as the final results.

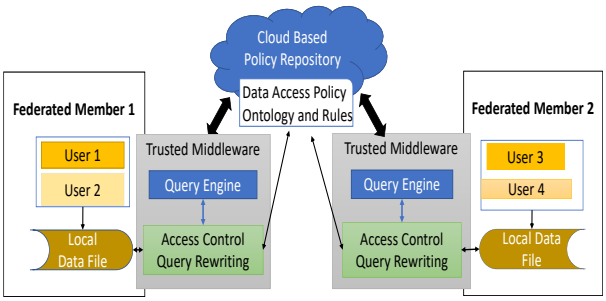

Figure 1. System Architecture for Situation-aware Access Control System

Overview of system architecture: Figure 1 illustrates the overall architecture of our system, which consists of federated members each with its own users and local data files.

To implement situation-aware access control, we first develop an ontology to help express access control rules for each member. Members can then define their access control rules using the ontology. These rules are stored in a cloud based policy repository for ease of access by all members. Each member can also store a local copy of the ontology if the connection is unreliable.

Data-as-a-Service and situation-aware access control are implemented in our architecture as a trusted middleware layer at each member. The middleware consists of a query engine as well as an access control query rewriting module. The access control query rewriting module enforces situation-aware access control rules by adding constraints derived from these rules to each query such that only results accessible by the user who asks the query are returned. The query engine at each member communicates with the query engines at other members and together they execute federated queries using data stored at multiple members.

At run time, a user of a federated member submits a query to the query engine. The access control query rewriting module adds access-control constraints to the query and rewrites the query using rules stored in the cloud based policy repository. The rewritten query is executed by the query engines of multiple members and the final results are returned to the user. Note that query rewriting ensures that the final results only contain data the user is allowed to access based on the situation, as defined by the rules.

Peer-to-Peer vs. Trusted Coordinator architecture: Our approach uses a peer-to-peer architecture where each member's middleware layer can directly query other members' middleware. To simplify the process to join our system, we assume that there is a known super-peer (e.g., in SAR missions, there is a SAR center) and each member contacts the super-peer to join the system. However, the super-peer is not involved in query processing.

An alternative architecture is to have a single trusted coordinator. In such an architecture, all queries are sent to the coordinator. The coordinator rewrites a query based on access control rules, and sends the query to the query engines of multiple members to get final results. The final

Figure 2. Example for rules RS1 and RS2

results are then sent to the member who initiates the query. However this architecture has a single point of failure and a bottleneck: the coordinator. In addition, most situation-aware access control rules contain conditions concerning the member who initiates the query, e.g., to verify that the user is the captain of a vessel that is indeed in distress. Using our peer-to-peer architecture, some fragments of the rewritten query will be executed locally at the member where the data resides. For example, the fragment to verify the role of the captain will be executed at the vessel. Using the trusted coordinator architecture, the coordinator needs to execute all fragments of query remotely, which often leads to worse performance due to higher network communication costs. Section V compares these two approaches experimentally.

III. Ontology and Rules

Situation-aware access control policies typically can be represented using the following components [1]:

- U: the set of users.
- R: the set of roles. R has a hierarchical structure.
- $UR \subseteq U \times R$: the assignment of users to roles.
- O: the set of data that can be shared.
- PU, PO: properties of data or users.
- SE: set of situation expressions typically on properties of users or objects (PU, PO).
- P: the set of permissions defined on O.
- $RP \subseteq R \times P$: the assignment of permissions to roles.
- $SEUR \subseteq 2^{SE} \times UR$, the set of situation-aware assignment of roles to users. 2^{SE} is power set of SE.
- $SERP \subseteq 2^{SE} \times RP$, the set of situation-aware role permission assignments.

We use the W3C Web Ontology Language (OWL) to define an ontology for maritime search and rescue and another ontology for contact tracing. Figure 3 and Figure 4 show major classes and relationships (red arrows) between classes in these ontologies, respectively. The major classes for Maritime SAR include the vessel in distress, organizations participating in the search and rescue mission, assets (e.g., medical equipment) owned by these organizations, the rescue coordination center, and their data to be shared in the system, roles, users, allowed operations (read or write), rescue missions in which they are involved, and tasks in the mission, etc. The designed ontology contains 14 classes and 20 relationships between these classes.

The ontology for contact tracing includes major classes such as person (being investigated), different types of data about the person such as EHR data, air travel data, hotel data, cruise travel data, users (e.g., contact tracers), roles, and organizations (e.g., air lines, cruise ships, hotels, health-care providers). The ontology contains 13 classes and 14 relationships. Many classes also have data properties. We also assume that in both ontologies the User class can have hasReadAccess or hasWriteAccess relationship to any other classes or properties of classes other than the role class.

The rules define the situation-aware assignment of permissions (RP and $SERP$) or roles ($SEUR$). The rules consist of conditions and consequences. Conditions are typically predicates on the situation or roles of users, and consequences are typically whether a user is allowed to carry out a certain operation on a certain type of data item or whether a certain situation occurs.

Sample Rules: The following are several SAR use case sample rules. These rules are written in a format slightly different from the standard OWL format to increase readability.

Rule RS1 defines a dynamic situation: two ships are within range if they are closer than 100km from each other. Rule RS2 shows situation-aware assignment of permissions in which the SAR coordinator can update the destination of search and rescue ships that are near a vessel in distress. Rule RS3 specifies that the captain of a vessel in distress can have access to the assets of a member in the SAR mission, and Rule RS4, specifies that the captain has access to the location of members in the SAR mission.

- **RS1**: (?V ns:has ?VL) $\wedge$ (?VL rdf:type ns:Location) $\wedge$ (?O ns:has ?L) $\wedge$ (?L rdf:type ns:Location) $\wedge$ EuclideanDistance(?VL, ?L) $<$ 100KM $\rightarrow$ (?O ns:isWithinRangeOf ?V)
- **RS2**: (?U ns:hasRole ns:SARCoordinator) $\wedge$ (?U ns:belongsTo ?S) $\wedge$ (?O ns:isMemberOf ?S) $\wedge$ (?O ns:has ?D) $\wedge$ (?D rdf:type ns:Destination) $\wedge$ (?V ns:hasStatus ns:Distressed) $\wedge$ (?O ns:isWithinRangeOf ?V) $\rightarrow$ (?U ns:hasWriteAccess ?D)
- **RS3**: (?U ns:belongsTo ?V) $\wedge$ (?U ns:hasRole ns:VesselCaptain) $\wedge$ (?V ns:contacts ?S) $\wedge$ (?V ns:hasStatus ns:Distressed) $\wedge$ (?O ns:isMemberOf ?S) $\wedge$ (?O ns:has ?A) $\wedge$(?A rdf:type ns:Asset) $\wedge$(?V ns:inMission ?M) $\wedge$(?O ns:inMission ?M) $\rightarrow$ (?U ns:hasReadAccess ?A)
- **RS4**: (?U ns:belongsTo ?V) $\wedge$(?U ns:hasRole ns:VesselCaptain) $\wedge$ (?V ns:contacts ?S) $\wedge$(?V ns:hasStatus ns:Distressed) $\wedge$(?O ns:isMemberOf ?S) $\wedge$(?O ns:has ?L) $\wedge$(?L rdf:type ns:Location) $\wedge$(?O ns:inMission ?M) $\wedge$(?V ns:inMission ?M) $\rightarrow$ (?U ns:hasReadAccess ?L)

According to rules RS1 and RS2, a user who is a SAR coordinator can update the destination of SAR Ship 2 in Figure 2 as it is within 100km of the ship in distress, but cannot update SAR Ship 1 as it is not in the range.

Below are two sample rules in the contact tracing case. Rule RC1 states that if a person A has status "Person Under

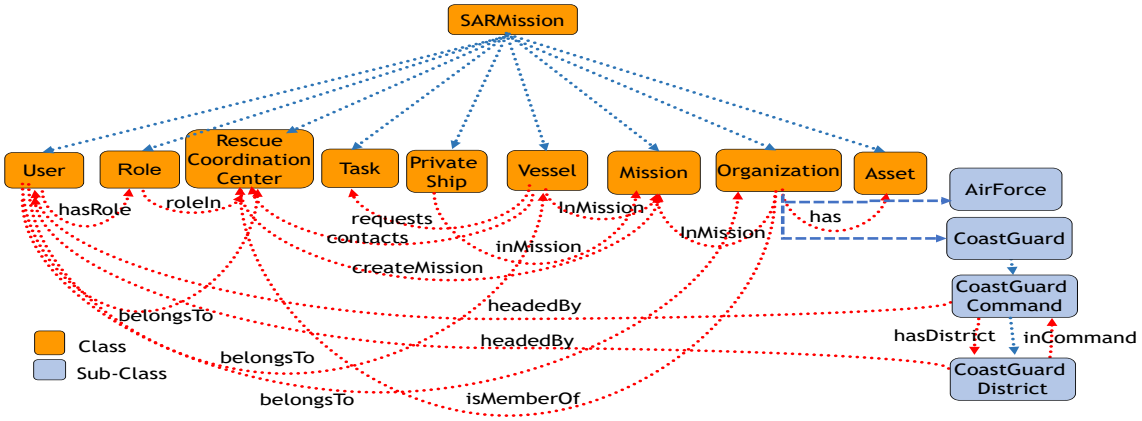

Figure 3. Ontology for Maritime SAR

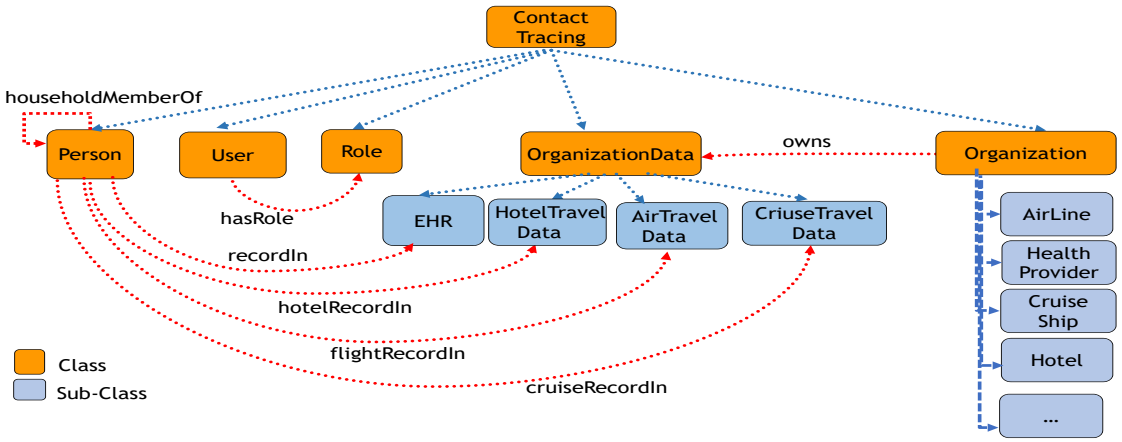

Figure 4. Ontology for Contact Tracing

Investigation" (PUI) (usually when this person is diagnosed with the disease), and a person B is a household member of person A, then B has "close contact" status. Rule RC2 states that a user with the role contact tracer can have access to EHR data about a person who has "close contact" status.

- **RC1**: (?PersonA td:status td:PUI)∧(?PersonB td:householdMemberOf ?PersonA) → (?PersonB td:status td:CloseContact)
- **RC2**: (?Person td:status td:CloseContact) ∧ (?Person td:recordIn ?EHR) ∧ (?User td:hasRole td:ContactTracer) ∧ (?HealthCare td:owns ?EHR) ∧ → (?User td:hasReadAccess ?EHR)

IV. Distributed Reasoning Framework Based on Query Rewriting

When a user asks a query in a federated data-as-a-service system, the system needs to distributively reason with situation-aware access control rules to only return results the user is allowed to see.

To accomplish this, we propose a query rewriting method, *Situation-Aware-Access-Control-Rewrite* (SAAC-Rewrite), to convert distributed reasoning into distributed query processing, which is easier to support in existing data-as-a-service systems. As shown in Algorithm 1, the algorithm consists of an offline pre-processing step, detailed in Algorithm 2 and an online query rewriting step, detailed in Algorithm 3. An early version of these algorithms was proposed in our preliminary work [11]. In this paper we have significantly extended Algorithm 3 to handle query rewriting in a distributed environment.

In the following discussion, Q is a SPARQL query, RS is the set of access control rules, u is the user issuing the query, and I is the set of rule conditions that need to be inferred (e.g., the isWithinRangeOf condition in rule RS1). We also assume that policy rules are not recursive.

Preprocessing: Algorithm 2 preprocesses the access control rules such that any derived predicate that appears in the condition part of a rule is replaced with predicates

Algorithm 1 : SAAC-Rewrite(RS, Q)

1: RS' =Preprocess(RS) {Rewrite the rule to remove derived conditions}
2: Rewrite-Online(RS', Q) {Rewrite the query online}

that are in the raw data, so there is no need for runtime inference. This essentially allows us to convert a distributed reasoning problem to a federated query problem. This is possible because we assume that access control rules are not recursive. Algorithm 2 works in a manner similar to backward chaining, i.e., by repeatedly replacing such a predicate p (line 3) with predicates in the condition part of a rule where p appears in the consequence (lines 4-5).

Algorithm 2 : Preprocess(RS)

1: **repeat**
2: **for** each $r = cond \rightarrow cons \in RS$ **do**
3: **for** each $p \in cond$ and $p \in I$ where I is set of derived predicates **do**
4: find rules $r'_1 = cd'_1 \rightarrow cs'_1$, $r'_2 = cd'_2 \rightarrow cs'_2$, ..., $r'_k = cd'_k \rightarrow cs'_k \in RS$ such that $\forall j, p \in cs'_j$
5: replace p in r with $cd'_1 \vee cd'_2 \vee \ldots \vee cd'_k$
6: **end for**
7: **end for**
8: **until** no more rules are rewritten

Preprocessing example: When Algorithm 2 is applied to the RS1 and RS2 rules introduced in Section III, the (?O ns:isWithinRangeOf ?V) condition in rule RS2, which is the same as the consequence in rule RS1, is replaced by the conditions in rule RS1.

Access Control Query Rewriting: Algorithm 3 shows our online query rewriting algorithm that enforces access control rules. Step 1 initializes NQ to store rewritten query and C to store the set of conditions to check permissions. In steps 2 to 3, for each returned variable v_i in SELECT clause, a new triple pattern t_i is added to the query to check that the user issuing the query has the correct permissions to access v_i. This is the key step in our proposed framework for situation-aware access control as constraints are added to the query to ensure that the end user only receives the results they have permission to see. The algorithm then checks for each rule whose consequence's predicate matches the predicate in t_i (lines 4-7) and calls a function VariableMatch to rewrite the rule such that the variables in the rule's conditions are replaced with matching variables or constants from the query (line 5). The function returns a set of conditions of the processed rule and adds them to C. Due to space constraints, we omit pseudo code for the VariableMatch function, but include an example below which illustrates how it works.

If multiple access control rules match the same triple pattern t_i, only one of them needs to be satisfied in order for the user to have access to the result. In line 8, GenAllQueries function generates all possible queries where each query has the same conditions as Q except that each of the added

Algorithm 3 Rewrite-Online(RS, Q)

1: NQ = $\emptyset$; $C = \emptyset$
2: **for** each variable v_i in SELECT **do**
3: add a triple $t_i = (u, hasReadAccess, ?v_i)$ to Q
4: **for** each rule $r \in RS$ whose consequence predicate matches t_i's predicate **do**
5: add C_{ir} =VariableMatch(r, Q, t_i) to C
6: **end for**
7: **end for**
8: $QS = GenAllQueries(Q, C)$
9: **for** each query $Q_j \in QS$ **do**
10: $P = \emptyset$; Q'_j = prefix and select clause of Q
11: **for** each triple pattern $q \in Q_j$ **do**
12: **for** each service endpoint p_q **do**
13: add (p_q, q) to P if p_q returns non empty result for q using SPARQL ASK query
14: **end for**
15: **end for**
16: sort items in P by endpoints and remove duplicates
17: **for** each service endpoint $p_q \in P$ **do**
18: **if** p_q is not local **then**
19: add SERVICE keyword at p_q to Q'_j
20: **end if**
21: add to Q'_j all triple patterns in P at p_q
22: **end for**
23: add Q'_j to NQ and add a UNION keyword if $j > 1$
24: **end for**
25: return NQ

triple t_i is replaced with triple patterns (conditions) returned by one matching rule. The final result is a union of results generated by these queries (line 23).

Some of the triple patterns in these queries need to be evaluated using a remote member's data. Lines 11 to 15 use SPARQL ASK queries to find the service end points that have non-empty results for each triple pattern in a generated query. Triple patterns belonging to the same endpoint are then placed in the same group and duplicated triple patterns are removed (line 16). If the endpoint is remote, the SERVICE keyword is added to get results from that endpoint (line 19). Finally, all queries generated in line 8 are unioned in line 23 and the resulting query NQ is returned (line 25).

Query rewriting example: Consider query QS_1 described in Section II. Suppose that QS_1 is issued by "John", the captain of a vessel in distress. Following Algorithm 3 line 3, once QS_1 is issued by "John", the algorithm first adds the triple pattern t_1 = `ns:John ns:hasReadAccess ?Result` to the query. Both rules RS3 and RS4 in Section III match the added triple pattern. Function VariableMatch is invoked to replace variables in rule RS3 first.

The VariableMatch function tries to match every triple in the rule with a triple pattern in the query (including t_1) based on their predicate. There are two cases: 1) the subject or object (s_r) of a triple in the rule is a variable so

it needs to be replaced with the subject or object (s_q) of the matching query triple; 2) s_r is a constant but the matching s_q is a variable so s_q needs to be bounded to s_r. The second case can be handled by using a BIND(s_r AS s_q) form and replacing s_q in other triple patterns in Q with s_r.

In the above example, the triple pattern `?U ns:hasReadAccess ?A` in rule RS3 matches triple pattern `ns:John ns:hasReadAccess ?Result` in the query, so the variable $?U$ is replaced with `ns:John`. Similarly $?A$ in the rule is replaced with $?Result$.

After such replacement, the conditions of rule RS3 will be returned. Similarly, the conditions of rule RS4 will be returned. At step 8, two queries will be generated, each containing conditions in one of these rules as well as triple patterns in the original query.

Algorithm 3 then uses SPARQL ASK queries to find the service end point matching each triple pattern. In our experimental setup (discussed in Section V), triple patterns with subject ?Organization or ?Result are matched using the remote service endpoint. The other triple patterns are matched locally. The algorithm repeats for the query generated from rule RS4 and unions the results from both rules in line 23. The final rewritten query is:

```
PREFIX ns: <http://www.sar.org/ns#>
SELECT DISTINCT  ?Result WHERE{{
ns:John ns:belongsTo ?V .
ns:John ns:hasRole ns:VesselCaptain .
?V ns:contacts ?SARCenter .
?V ns:hasStatus ns:Distressed.
?V ns:inMission ?M .
SERVICE <http://192.168.56.103:3030/AF> {
?Organization ns:has ?Result .
?Result  rdf:type   ns:Location .
?Organization ns:isMemberOf  ?SARCenter .
?Organization ns:inMission ?M .
} } UNION {
ns:John ns:belongsTo ?Vessel .
ns:John ns:hasRole ns:VesselCaptain .
?Vessel ns:contacts ?SARCenter .
?Vessel ns:hasStatus ns:Distressed .
?Vessel ns:inMission ?M .
SERVICE <http://192.168.56.103:3030/AF> {
?Organization ns:has ?Result .
?Result rdf:type ns:Asset .
?Organization ns:isMemberOf ?SARCenter .
?Organization ns:inMission ?M . } }}
```

Complexity: Let n_r be the number of rules, n_p be the average number of conditions in a rule, n_t be the number of triple patterns in query Q, n_v be the number of variables in SELECT and n_s be the number of service endpoints. The function VariableMatch is called $O(n_v n_r)$ times. Each call costs $O(n_p n_t)$ as the function matches every query triple pattern with a rule condition. In line 8, $O(n_r^{n_v})$ queries can be generated because each added triple can match up to n_r rules. For each generated query, $O((n_p + n_t)n_s)$ ASK queries are executed. Sorting the items at line 16 of Algorithm 3 costs $O((n_p + n_t)n_s \log (n_p + n_t)n_s)$. The total cost of Algorithm 3 is $O(n_v n_p n_t n_r + n_r^{n_v}(n_p + n_t)n_s \log (n_p + n_t)n_s)$. Note that n_v is typically quite small. For example, in the Lehigh University Benchmark [15], n_v ranges from 1 to 4 in the 14 test queries and the average of n_v is 1.78.

Correctness: Since the rewritten query is executed in a trusted middleware in our architecture and users do not see intermediate results, we only need to prove that our query rewriting Algorithm 3 satisfies the following two properties described in [16]: soundness (the rewritten query only returns results in the initial query) and maximality (the rewritten query returns as much information as possible).

The soundness of Algorithm 3 is straightforward as the rewritten query still contains all existing triple patterns (conditions) in the initial query, so any result returned by the rewritten query also satisfies the initial query.

To prove that our algorithm is maximal, we need to show that any result returned by the initial query, but not by the rewritten query, must be a result that the user is not authorized to see.

Line 3 of Algorithm 3 adds triple patterns to check whether the user has access to each variable in SELECT. At line 8, the algorithm generates a number of queries where each is identical to the original query except that each triple pattern added in line 3 is replaced with conditions of a matching access control rule. Lines 11 to 22 of Algorithm 3 distribute the rewritten query to different members in the system. The final result is union of all generated queries.

Let C_Q be conditions in original query. Let RS_i be the set of rules matching a added triple t_i for variable v_i in SELECT. Let C_{ij} be conditions of rule $r_{ij} \in RS_i$. The result of each generated query satisfies conditions $(\wedge_{i=1}^{n_v} C_{ij}) \wedge C_Q$, for some $(r_{i1}, \ldots, r_{in_v}) \in RS_1 \times \ldots RS_{n_v}$. The rewritten query returns union of all possible queries. So a result returned by the rewritten query must satisfy $\vee_{(r_{i1},\ldots,r_{in_v}) \in RS_1 \times \ldots RS_{n_v}} ((\wedge_{i=1}^{n_v} C_{ij}) \wedge C_Q)$, which equals $(\vee_{(r_{i1},\ldots,r_{in_v}) \in RS_1 \times \ldots RS_{n_v}} (\wedge_{i=1}^{n_v} C_{ij})) \wedge C_Q$.

So any result returned by initial query satisfies C_Q, but not the rewritten query, must not satisfy $\vee_{(r_{i1},\ldots,r_{in_v}) \in RS_1 \times \ldots RS_{n_v}} (\wedge_{i=1}^{n_v} C_{ij})$. Thus the result does not satisfy access control conditions $\wedge_{i=1}^{n_v} C_{ij}$ checked by any generated query. However if the user is authorized to see the result, for each variable v_i in SELECT, there must be some access control rule whose conditions (C_{ij}) are satisfied. This contradicts with the above. So user is not authorized to see the result.

V. Implementation and Evaluation

We have implemented a system in which the ontology and rules are developed using Protégé [17]. We used Apache

Jena Fuseki [18] to store data as RDF triples and provide a web service interface to support SPARQL 1.1. queries over multiple endpoints. Each member of the federated system was implemented as an Apache Jena Fuseki server. Apache Jena Fuseki only supports local reasoning (both data and rules are at the same endpoint). In our implementation, the rules are stored in the cloud but data are stored at different members. We implemented our middleware responsible for enforcing access control rules using query rewriting.

A. Validation of Ontology and Rules

We validated the Maritime SAR proposed ontology with SAR domain experts from the U.S. Coast Guard and U.S. Navy[1]. The ontology for contact tracing has been validated by a public health expert[2]. These experts confirmed that the domain assumptions, classes, properties, and relationships defined by our ontology support the extraction, aggregation, and sharing of relevant information in each use case.

B. Experiment Setup

We assume that for each use case there are multiple members and each member has its own data but shares the same ontology. We created five Ubuntu virtual machines to simulate different members. Each VM has 3 GB of memory and 80GB of hard disk. Each VM runs a Jena Fuseki Server as well as middleware implementing our query rewriting algorithm. We varied the number of endpoints in the experiments. We also simulated different network delays, ranging from no delay, 50 ms (typical in a 4G network), 100 ms (3G network), to 500 ms (typical in satellite communication).

Methods: We compare the performance of the following algorithms.

1) SAAC-Rewrite: this is our proposed algorithm using the peer-to-peer architecture shown in Figure 1. Query rewriting occurs at the member issuing the query. Parts of the rewritten query are executed locally and other parts are executed remotely.
2) Trusted-Coordinator: this is the alternative solution discussed in Section II, where all queries are first sent to a trusted coordinator. The trusted coordinator runs a similar query rewriting algorithm and sends parts of the rewritten query to different service end points. It then puts together final results and sends them back to the member who issued the query. Since data are distributed at different members, all parts of the query are executed remotely.

[1]Interviews with: Nautical Science Instructor (search and rescue domain expert), U. S. Naval Academy, Feb 2019; retired U. S. Naval Officer, March 2019; Chief, Incident Management Division, Sector Maryland, U. S. Coast Guard, April 2019.

[2]Interviews with a public health expert at Johns Hopkins University, May 2020.

Data set	Member 1 (Vessel in Distress)	Member 2 (Coast guard)	Member 3 (Air Force)
Small	52	96	86
Medium	520	960	860
Large	5200	9600	8600

Table I
TOTAL NUMBER OF TRIPLES IN SMALL, MEDIUM AND LARGE DATA SETS FOR SAR

Data Set	Member 1 (Contact Tracer)	Member 2 (Healthcare Provider)	Member 3 (Airline)
Small	66	53	61
Medium	660	530	610
Large	6600	5300	6100

Table II
TOTAL NUMBER OF TRIPLES IN SMALL, MEDIUM AND LARGE DATA SETS FOR CONTACT TRACING

3) Centralized: this is the case when all data are stored at one place. The queries are still rewritten to verify situation-aware access control conditions but without the need to query remote nodes.

Datasets: It is difficult to obtain publicly available data sets for the two use cases we consider. We generated three synthetic data sets for each use case: Small, Medium, and Large, based on the created ontology. We tried to model the synthetic data as realistically as possible to the respective use case. For the maritime SAR use case, we generated data based on information from a large scale multinational maritime search and rescue exercise conducted in 2016 [19]. For the contact tracing use case, we generated data based on information available from the US Centers for Disease Control and Prevention [20].

Table I shows the number of triples in the SAR data and Table II shows the numbers in the contact tracing data. The Medium data set is scaled by a factor of 10 over the Small data set. The Large data set is scaled by a factor of 100 over the Small data set.

In the SAR data sets, there are three members: a vessel in distress, a member of the U.S. Coast Guard, and a member of the U.S. Air Force. Each member stores data related to itself. E.g., vessel in distress member stores data about passengers and the incident.

Contact tracing data sets are distributed on three members: a contact tracer who has data about people being diagnosed with the disease, a healthcare provider which owns EHR data of patients, and an airline which owns air travel data.

Queries: For each data set we used two queries in the experiments. For SAR data sets, query QS_1 is the same as the example query in Section IV where the captain of a vessel in distress asks for data about organizations belonging to the SAR center that oversees the rescue mission.

Query QS_2 is asked by the coordinator of an organization participating in the rescue mission to get data about the

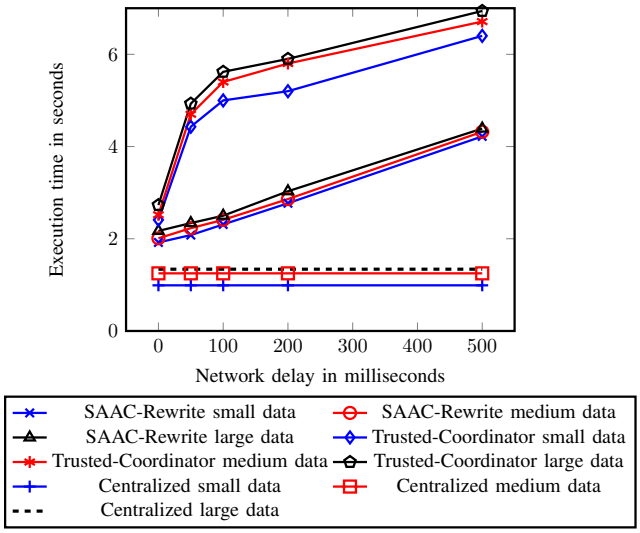

Figure 5. Execution time of Query QS_1 when varying network delay

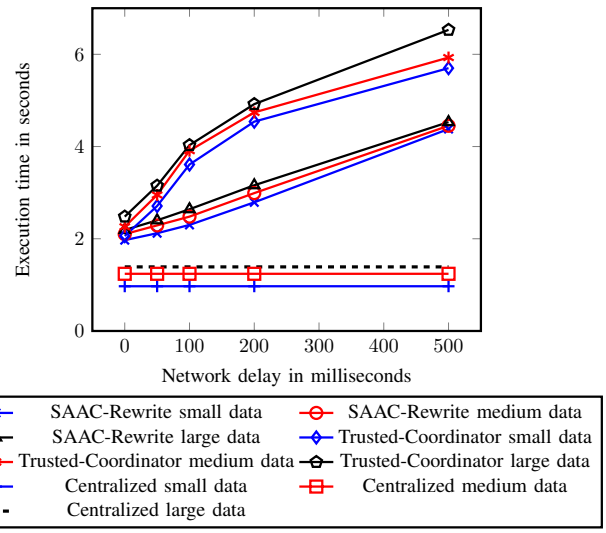

Figure 6. Execution time of Query QS_2 when varying network delay

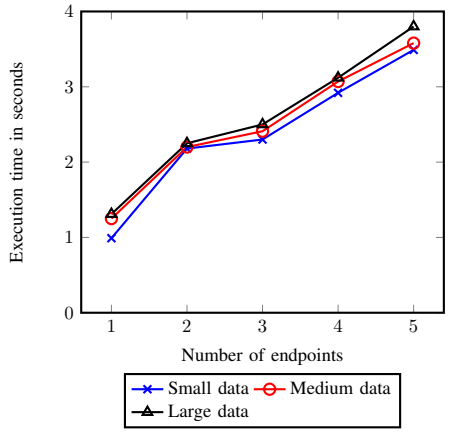

Figure 7. Execution time of Query QS_1 using SAAC-Rewrite when varying number of endpoints at 100ms delay.

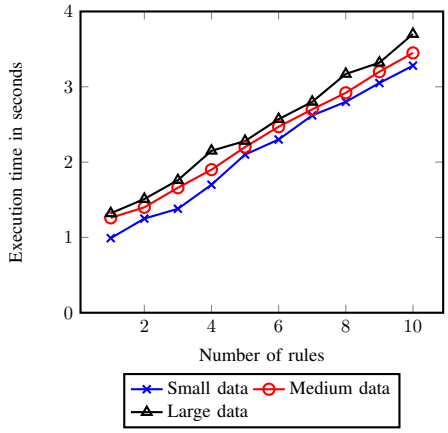

Figure 8. Execution time of Query QS_1 using SAAC-Rewrite when varying number of rules with 100 ms network delay

Data	# of results for QS_1	# of results for QS_2
Small	4	6
Medium	17	21
Large	167	172

Table III
SIZE OF QUERY RESULTS

vessel that asked for help:

```
QS2: PREFIX ns: <http://www.sar.org/ns#>
SELECT  ?Result
WHERE {
?Vessel ns:contacts ?SARCenter .
?Vessel ns:has ?Result . }
```

The rewritten query includes additional conditions from access control rules such as whether the user is coordinator of the SAR mission, and that the data is accessible by coordinator. For example, general passenger data of the vessel will be accessible, but not nationality of passengers (which may discourage passengers from seeking help).

For the contact tracing data sets, we used two queries. QC_1 is asked by a contact tracer to return the electronic health data and status of people where the EHR data is at a healthcare provider's site.

```
QC1: PREFIX td: <http://www.cdc.gov/td#>
SELECT  ?Person ?status ?EHR
WHERE {
?Person rdfs:status ?status .
?Person  td:recordIn  ?EHR .};
```

According to the access control rules, status and person are accessible by any users but only EHR data for people with the status PUI or with the status "close contact" are accessible by a contract tracer. So the rewritten query will check these conditions at the contract tracer site. The query also needs to retrieve EHR data for qualified people from healthcare provider's site.

QC_2 is asked by a contract tracer John to get air travel data (say passenger list) related to people under investigation from an airline.

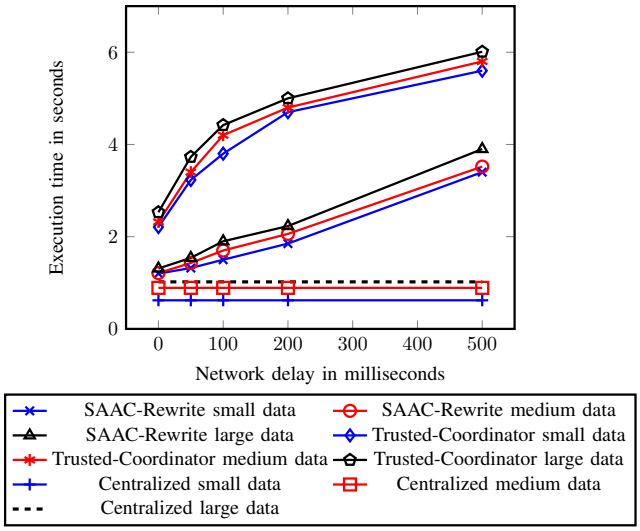

Figure 9. Execution time of Query QC_1 when varying network delay.

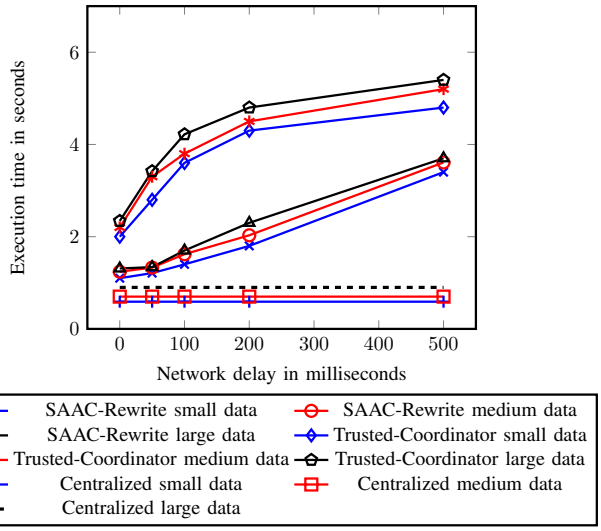

Figure 10. Execution time of Query QC_2 when varying network delay.

```
QC2: PREFIX td: <http://www.cdc.gov/td#>
SELECT ?Person ?Status ?AirtravelData
WHERE {
?Person rdfs:status ?Status.
?person td:flightRecordIn ?AirtravelData.}
```

According to access control rules, only users with contact tracer or case investigator roles can access air travel data about flights where a person with PUI status is on that flight. The rewritten query is shown below:

```
PREFIX td:   <http://www.cdc.gov/td#>
SELECT ?Person ?Status ?AirtravelData
WHERE {{?Person td:status  td:PUI.
 td:John td:hasRole td:ContactTracer.
 BIND (td:PUI AS ?Status)
SERVICE <http://192.168.56.103:3030/Trace>
{?Person  td:flightRecordIn ?AirtravelData.
?Airline  td:owns ?AirtravelData.}
} UNION { ?Person rdfs:status td:PUI.
td:John  td:hasRole   td:CaseInvestigator.
BIND (td:PUI AS ?Status)
SERVICE <http://192.168.56.103:3030/Trace>
{?Person td:flightRecordIn ?AirtravelData .
?Airline  td:owns ?AirtravelData.}}}
```

We used 16 rules in the experiment for the SAR case, where query QS_1 and QS_2 each matched 8 rules. For the contact tracing use case we used 8 rules where query QC_1 matched 3 rules and QC_2 matched 2 rules.

C. Results

Correctness of Query Rewriting Algorithm: We compared returned query results of the proposed SAAC rewriting algorithm to the centralized solution. Table III reports the size of the result set when executing the queries for various data sets for the SAR case. The query results of our algorithm are identical to the centralized solution, the experiment thus validating our theoretical result that our proposed distributed query rewriting algorithm is correct. The results for contact tracing are similar and omitted.

Results when varying network delay: Figure 5 shows the execution time of QS_1 and Figure 6 shows the execution time of QS_2 where we vary the network delay from no delay to 50ms, 100ms, 200ms, and 500ms for the three data set sizes, each distributed across three endpoints for the SAR use case. Figure 9 and Figure 10 show the results for QC_1 and QC_2 over three data sets for the contact tracing case with three endpoints but only data from two endpoints (member 1 and 2 for QC_1 and member 1 and 3 for QC_2) are needed to answer each query.

The execution time includes both query rewriting time and time to execute the rewritten query. The time is recorded in seconds. The query rewriting time is quite small compared to the execution time of the rewritten query.

The execution time of the centralized case (i.e., when all data are in one place) and Trusted-Coordinator are also reported. The execution time of the centralized case is flat because all data are stored at one place, so network delay has no impact on execution time.

The results show that the execution time of the SAAC-Rewrite method increases almost linearly with network delay. Although its execution time is higher than that of the centralized case, it is still acceptable, e.g., at only 4.39 seconds on the Large SAR data set with 500 ms delay.

The execution time of the Trusted-Coordinator approach is in the range of 30% to 110% higher than that of SAAC-Rewrite on SAR data sets and is in the range of 50% to 150% higher than that of SAAC-Rewrite on contact tracing data sets, mainly because Trusted-Coordinator has to execute all parts of a query remotely while SAAC-Rewrite executes part of the query locally. The local part also always contains

user information (e.g., to verify the role of the user), so the result size of the local part of the query is very small. Since the results from remote queries need to be joined with local results, this also reduces overall execution time.

Our results also show that when data size increases, execution time of SAAC-Rewrite increases just slightly. The reason is that each query is asked by a specific user. To check whether the user has access to the results, triples with subject or object equal to the user are added to the rewritten query. For example in QS_2, triples "ns:Peter ns:hasRole ns:SARCoordinator" and "ns:Peter ns:belongsTo ?Organization" are added during rewriting. This makes the query more selective and thus more efficient to execute because not many intermediate results need to be sent over the network.

Results when varying number of endpoints: Figure 7 shows execution time of query QS_1 using SAAC-Rewrite when we vary the number of endpoints from 1 to 5 with 100 ms network delay for the SAR use case. The results for QS_2 and the contact tracing case are similar and omitted.

The case with one endpoint is the centralized case. When there are two endpoints, we include Member 1 and Member 2 in Table I. When there are four or five endpoints, we generate data similar to Member 2 for Members 4 and 5. The results indicate that execution time of SAAC-Rewrite increases roughly linearly with the number of endpoints for two reasons. First, each endpoint has its own set of data so when the number of endpoints increases, more data needs to be queried and this increases execution time. Second, since there are more data, more intermediate and final results need to be transferred between endpoints, which also increases execution time.

Results when varying number of rules: Figure 8 shows execution of QS_1 using SAAC-Rewrite when we vary the number of rules matching the query from 1 to 10 on various data sets with three endpoints and 100 ms network delay. The results show that the execution time of the proposed SAAC-Rewrite method increases linearly with the number of rules involved. The results for other queries are similar and omitted.

VI. RELATED WORK

Yau et al. [1] proposed a situation-aware access control framework for distributed settings with a model for representing access control rules. We used a similar model in our approach. However, instead of query rewriting, they assume that users explicitly request access to individual objects and then their solution decides whether a user is allowed to have access. This will not be efficient for cases in which a user asks a query that may return many thousands of objects.

There has been work on using semantic web technologies to enforce access control or privacy preferences. Beimel and Peleg [2] propose a situation-aware access control model based on OWL ontology and SWRL rules. A similar semantic-based approach was proposed by Sun et al. [7] and applied to e-Healthcare. Kayes et al. [8] used an ontology-based solution to represent purpose-oriented situations and use that in access control of software services. Oulmakhzoune et al. [9] used ontologies and query rewriting to enforce privacy preferences for data stored at a single place. Padia et al. [10] applied a query rewriting approach to enforce fine-grained access control to RDF data stored at a single place. However, none of these works considers distributed environments, where distributed reasoning and efficiency are two major challenges. Our work addresses these two challenges in a distributed environment.

Query rewriting has been used to answer queries over data with different schema or ontologies. Two popular techniques for relational data are Local-as-View (LAV) and Global-as-View (GAV). LAV represents the local schema as a view of the mediated global schema, and GAV represents the global schema as a view of local schema [4]. Thieblin et al. [5] proposed a rewriting method when there exists 1 to M mappings between two ontologies. Venetis et al. [6] proposed a method that can expand an existing rewritten query when the ontology is expanded. However, none of these efforts considers access control in a distributed environment which is addressed by our work. Our work currently does not consider multiple ontologies but we can certainly extend our work to multiple ontologies in the future.

In terms of application domains, Bruggemann et al. propose an ontology based approach to track vessel movement and detect abnormal behavior [21], and an ontology for surveillance of COVID-19 has been proposed [22].

VII. CONCLUSION AND FUTURE WORK

Existing work supporting distributed reasoning requires special systems, making such reasoning hard to deploy in current systems. Efficiency can also be a challenge in a resource limited environment such as maritime SAR. In this paper we propose an approach to supporting situation-aware access control in federated Data-as-a-Service systems using semantic reasoning that can be easily integrated with existing systems through query rewriting. Our peer-to-peer architecture allows the rewritten query fragments to be evaluated where the data resides, reducing the communication overhead.

For future work, we plan to extend our proposed ontologies and use real data to evaluate our approach. We also plan to develop methods for distributed trust management, as data sharing in federated systems depends on trust between members.

ACKNOWLEDGEMENT

This work was partially supported by Office of Naval Research grant# N00014-18-1-2452.

REFERENCES

[1] S. S. Yau, Y. Yao, and V. Banga, "Situation-aware access control for service-oriented autonomous decentralized systems," in *Autonomous Decentralized Systems*. IEEE, 2005, pp. 17–24.

[2] D. Beimel and M. Peleg, "Using owl and swrl to represent and reason with situation-based access control policies," *Data & Knowledge Engineering*, vol. 70, no. 6, pp. 596–615, 2011.

[3] E. Oren, S. Kotoulas, G. Anadiotis, R. Siebes, A. ten Teije, and F. van Harmelen, "Marvin: Distributed reasoning over large-scale semantic web data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 4, pp. 305–316, 2009.

[4] A. Halevy, A. Rajaraman, and J. Ordille, "Data integration: The teenage years," in *Proceedings of the 32nd international conference on Very large data bases*, 2006, pp. 9–16.

[5] É. Thiéblin, F. Amarger, O. Haemmerlé, N. Hernandez, and C. T. dos Santos, "Rewriting select sparql queries from 1: n complex correspondences." in *The 11th International Workshop on Ontology Matching*, 2016, pp. 49–60.

[6] T. Venetis, G. Stoilos, and G. Stamou, "Query rewriting under query extensions for owl 2 ql ontologies," in *The 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2011)*, 2011, p. 59.

[7] L. Sun, H. Wang, J. Yong, and G. Wu, "Semantic access control for cloud computing based on e-healthcare," in *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 2012, pp. 512–518.

[8] A. Kayes, J. Han, and A. Colman, "An ontological framework for situation-aware access control of software services," *Information Systems*, vol. 53, pp. 253–277, 2015.

[9] S. Oulmakhzoune, N. Cuppens-Boulahia, F. Cuppens, and S. Morucci, "Privacy policy preferences enforced by sparql query rewriting," in *2012 Seventh International Conference on Availability, Reliability and Security*, 2012, pp. 335–342.

[10] A. Padia, T. Finin, A. Joshi *et al.*, "Attribute-based fine grained access control for triple stores," in *3rd Society, Privacy and the Semantic Web-Policy and Technology workshop, 14th International Semantic Web Conference*, 2015.

[11] S. Oni, Z. Chen, A. Crainiceanu, K. Joshi, and D. Needham, "Situation-aware access control in federated data-as-a-service for maritime search and rescue," in *2019 IEEE International Conference on Services Computing (SCC)*, 2019, pp. 228–230.

[12] O. Lassila and R. R. Swick, "Resource description framework (rdf) model and syntax specification," *WWW Consortium*, 1999.

[13] D. L. McGuinness, F. Van Harmelen *et al.*, "Owl web ontology language overview," *W3C recommendation*, vol. 10, no. 10, 2004.

[14] The W3C SPARQL Working Group, "Sparql 1.1 overview," March 2013, accessed June 2, 2020. [Online]. Available: https://www.w3.org/TR/sparql11-overview/

[15] Y. Guo, Z. Pan, and J. Heflin, "Lubm: A benchmark for owl knowledge base systems," *Web Semantics*, vol. 3, no. 2-3, pp. 158–182, Oct. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2005.06.005

[16] Q. Wang, T. Yu, N. Li, J. Lobo, E. Bertino, K. Irwin, and J.-W. Byun, "On the correctness criteria of fine-grained access control in relational databases," in *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 555–566.

[17] M. A. Musen, "The protégé project: A look back and a look forward," *AI Matters*, vol. 1, no. 4, p. 4–12, Jun. 2015.

[18] Apache Jena, "Apache jena: A free and open source java framework for building semantic web and linked data applications," accessed June 2, 2020. [Online]. Available: https://jena.apache.org/

[19] Crystal Cruises, the Canadian Coast Guard, Transport Canada, and the U. S. Department of Defense (U.S. Air Force, and U.S. Coast Guard), "Northwest Passage (NWP 16) 2016 Exercise – After Action Report," July 2016, accessed Feb 7, 2019. [Online]. Available: https://www.hsdl.org/?view&did=802138

[20] Centers for Disease Control and Prevention, "Contact Tracing," May 2020, accessed May 12, 2020. [Online]. Available: https://www.cdc.gov/coronavirus/2019-ncov/php/open-america/contact-tracing-resources.html

[21] S. Brüggemann, K. Bereta, G. Xiao, and M. Koubarakis, "Ontology-based data access for maritime security," in *International Semantic Web Conference*. Springer, 2016, pp. 741–757.

[22] H. Liyanage, S. de Lusignan, and J. Williams, "Covid-19 surveillance ontology," March 2020, accessed May 26, 2020. [Online]. Available: https://bioportal.bioontology.org/ontologies/COVID19