Ontology-based data integration in a distributed context of coalition air missions

Karima Ennaoui, Mathieu Faivre, Shahriar Hassan, Christophe Rey, Lauren Dargent, Hervé Girod, Engelbert Mephu Nguifo

Ontology-based data integration in a distributed context of coalition air missions

Karima Ennaoui*, Mathieu Faivre*, Md Shahriar Hassan*, Christophe Rey*, Lauren Dargent**, Hervé Girod**, Engelbert Mephu Nguifo*

* Université Clermont Auvergne, LIMOS, UMR UCA-CNRS 6158
** Dassault Aviation, Saint-Cloud, France

Abstract. The IBC (Knowledge Base Integration) project addresses an issue of ontology-based data integration. It aims at combining data residing in different actors (aircraft, drone, satellite...) during an air mission scenario and providing users with a unified view of all available data, in a communication constrained environment. We describe the solution we have implemented based on mediation. We use rule languages to process queries using an OWL2 domain ontology and RDF triples to store data. We also give a performance analysis of our prototype.

1 Introduction

Within the MMT (Man Machine Teaming [1]) program launched by DGA (Direction Générale de l'Armement) and animated by Dassault Aviation and Thales, the IBC (Knowledge Base Integration) project is part of the "Virtual Assistant and Smart Cockpit" axis and focuses on the management of multiple knowledge and databases in a non-centralized environment. The study is about accessing and exploiting data from distributed platforms during coalition air operations. In the field of ontology-based data access (OBDA), this is the issue of ontological mediated query answering (OMQA) [4] in a distributed context, i.e. how to obtain answers to a query built using many data sources through the use of an ontology in a distributed environment. In the military air mission context, user queries are asked by actors involved in an air mission (fighter jets, drones, satellites, ...). Each actor can be viewed as a data source. The system must face the constraints of being able to generate real time answers to queries, with possible communication interruptions (either voluntarily or involuntarily) and limited storage capacities. In this paper, we propose a solution, its theoretical principle and its implementation.

In section 2, we present the architecture, followed by its components in section 3. Sections 4 and 5 give insights on performances and related works. We then conclude.

1. Cf. `https://man-machine-teaming.com`

2 Mediator crowd architecture

The proposed distributed architecture is a mediator crowd. It is a special mediation in which each actor is a mediator and is able to process queries with its own data and send subqueries, if needed, to other data sources to complete the answers. Classically, mediation, a.k.a. virtual data integration, does not require to materialize (i.e. copy) data from an actor to another one in order for the latter to answer queries using data of the former (cf. chapter 9 of [2]). Figure 1 illustrates this.

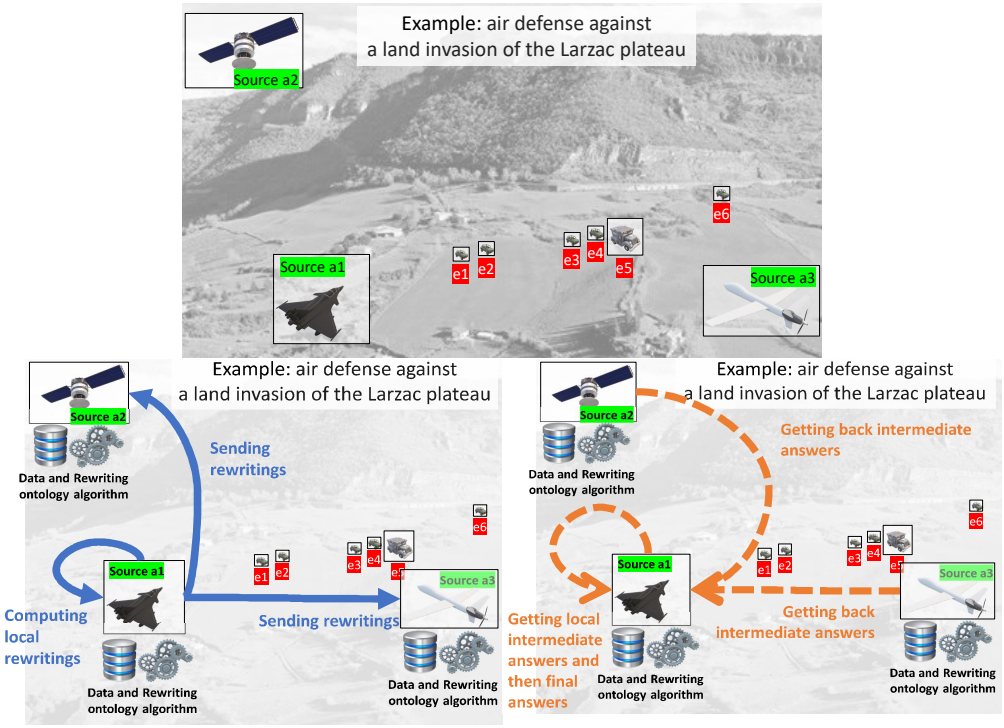

FIG. 1: Mediator crowd architecture on the example of a fictitious air mission with 3 actors (aircraft a1, satellite a2 and drone a3) who defend the Larzac plateau against a invasion led by 6 enemies (e1 to e6). On the bottom left, a1 sends the query rewritings to all actors. On the bottom right, a1 gets back results and computes the answers.

In figure 1, assuming a1, a2, and a3 share the same air mission ontology, but have their own data, the issue is the following: how can a1 answer queries using the ontology, its own data and also a2's and a3's data? An example of a query could be to identify a1's enemies. As illustrated in figure 1, the principle of a mediation crowd is the following. Thanks to the mediator components (see figure 2), a1 first rewrites the query into subqueries, called rewritings, each one corresponding to some distant actor. Then these rewritings are sent to the corresponding actors (including a1 itself). All actors processes their rewritings, thereafter sending the intermediate results to a1. At last, a1 builds the final answers using rewriting results from all actors.

3 Mediator components

As shown in figure 2, we now detail mediator components: the ontology, containing the domain knowledge and the mappings, facts (i.e. data) specific to each source and stored as RDF triples in RDF triple stores, and the query rewriting algorithm.

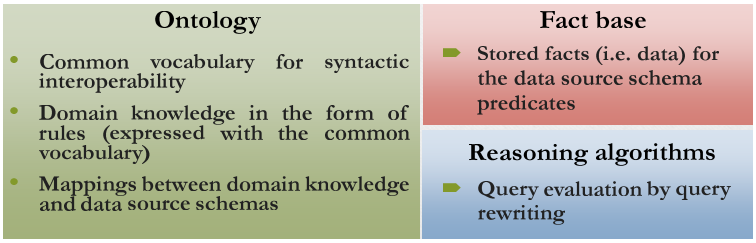

FIG. 2: The components of each mediator.

3.1 Ontology and mappings

The chosen mediation approach provides actors with an ontology describing the military air mission domain, giving a common terminology of concepts and properties such as "mission plan", "is enemy of" or "detectable by radar" and axioms describing some of their relationships. Actors use these to express their queries. The used modeling language is OWL2 [2], as it is the reference language in ontology modeling. To be used in the rewriting process, OWL2-modeled knowledge is then translated into so-called skolemized rules (see section 3.2)). In addition to the domain ontology, each actor has its own vocabulary, called "data source schema", to structure the storage of its own data.

Mappings are existential rules (see section 3.2) of the form $Q_O \leftarrow Q_S$ that link a query Q_S expressed on the data sources schemas to one Q_O expressed on the ontology vocabulary. This is called a global and local as view (GLAV) mapping in the data integration literature (see chapter 9 of [2] for example). Without loss of expressivity, mappings can be expressed as rules of the form : $P_O \leftarrow P_1, ..., P_k$, where P_O is a triple over the ontology and $P_1, ..., P_k$ is a conjunction of triples over data sources predicates (i.e. data source schema elements). Each triple has one of the following form: (i) ($?x$ ibc:pred $?y$) with x and y variables and *pred* a binary predicate (a property); (ii) ($?x$ rdf:type *pred*) with x a variable and *pred* a unary predicate (a class); (iii) ($?x$ ibc:pred Y) or (Y $ibc : pred$ $?x$) with x a variable, Y a constant and *pred* a binary predicate.

In a mapping, P_is can refer to the same data source schema or to different ones. Mappings are essential in the rewriting process (see section 3.2) in order for a query to be rewritten using only data source predicates. The ontology and the mapping associated to the example of figure 1 (once translated as rules) are given now [3] (ontology rules R1 to R6 and mappings M1 to M7):

2. See `https://www.w3.org/TR/owl2-overview/`.

3. We precise here that the ontology shown here is a toy one, for explanation purposes. The real air mission ontology we have used has been designed by Dassault Aviation and is called SITAC.

R1: $danger(f1(X,Y)) \leftarrow isDetectableBy(X,Y), isEnemyOf(Y,X)$.
R2: $putInDangerBy(X, f1(X,Y)) \leftarrow isDetectableBy(X,Y), isEnemyOf(Y,X)$.
R3: $isEnemyOf(X,Y) \leftarrow ally(X), enemy(Y)$.
R4: $isEnemyOf(X,Y) \leftarrow isEnemyOf(Y,X)$.
R5: $isRadarDetectableBy(X,Y) \leftarrow isRadarDetectableBy(X,Z), isRadarDetectableBy(Z,Y)$.
R6: $isDetectableBy(X,Y) \leftarrow isRadarDetectableBy(X,Y)$.
M1: $enemy(X) \leftarrow a_1\_enemy(X), a_2\_enemy(X), a_3\_enemy(X)$.
M2: $isRadarDetectableBy(X,Y) \leftarrow a_1\_isRadarDetectableBy(X,Y)$.
M3: $ally(X) \leftarrow a_1\_ally(X)$.
M4: $isRadarDetectableBy(X,Y) \leftarrow a_2\_isRadarDetectableBy(X,Y)$.
M5: $ally(X) \leftarrow a_2\_ally(X)$.
M6: $isRadarDetectableBy(X,Y) \leftarrow a_3\_isRadarDetectableBy(X,Y)$.
M7: $ally(X) \leftarrow a_3\_ally(X)$.

R1 and R2 say that if X is radar detectable by Y, and Y is an enemy of X, then (R1) there exists something which depends on both X and Y that is a danger, and (R2) X is put in danger by something which depends on both X and Y. R3 says X is an enemy of Y if X is an ally and Y an enemy. R4 defines the $isEnemyOf$ predicate to be symetric. R5 defines $isRadarDetectableBy$ to be transitive. R6 says $isRadarDetectableBy$ is a sub-property of $isDetectableBy$. Mapping M1 defines X as an enemy if X is listed as an enemy for actors a_1, a_2 and a_3. M2, M4 and M6 say that being radar detectable by something is the same notion for each actor as for the ontology. M3, M5 and M7 say that being an ally is the same notion for each actor as for the ontology.

3.2 Theoretical framework and query processing algorithm

We have chosen the knowledge representation and reasoning formalism called "existential rules" (noted e.r., a.k.a. datalog$^{\pm}$ [5, 9]) to be the theoretical framework of our proposal. Indeed: (i) e.r. are the main up-to-date theoretical framework for the study of OBDA issues, with many important theoretical results [4] and existing reasoning algorithms [7]; (ii) e.r. generalize the main profiles EL, QL and RL of OWL2 [4] which allows to benefit from the well-known modeling features of OWL2, and also, after a translation step, from the theoretical and practical results available for e.r.; (iii) e.r. allow the expression of more general joins than the main profiles of OWL2, of more cases of existential variables in the head of rules, and also allow recursive knowledge to be defined in a very natural way; (iv) rule-based formalisms have proven to be relevant for query evaluation, with or without ontology (mappings and queries are traditionally represented as rules in the data integration field), and e.r. generalize conjunctive queries (which are equivalent to SELECT FROM WHERE queries in SQL, or SPARQL Basic Graph Pattern queries); (v) e.r. can also be equivalently translated as definite clauses in order to process ontologies as logic programs. E.g. in section 3.1, rules R1 to R6 and mappings M1 to M7 constitute a logic program. This is of great value since research in logic programming (noted l.p.) has achieved a few very optimized interpreters that can be used to reason with ontologies.

4. EL, QL and RL profiles are restrictions of OWL2 allowing decidable and tractable reasoning.

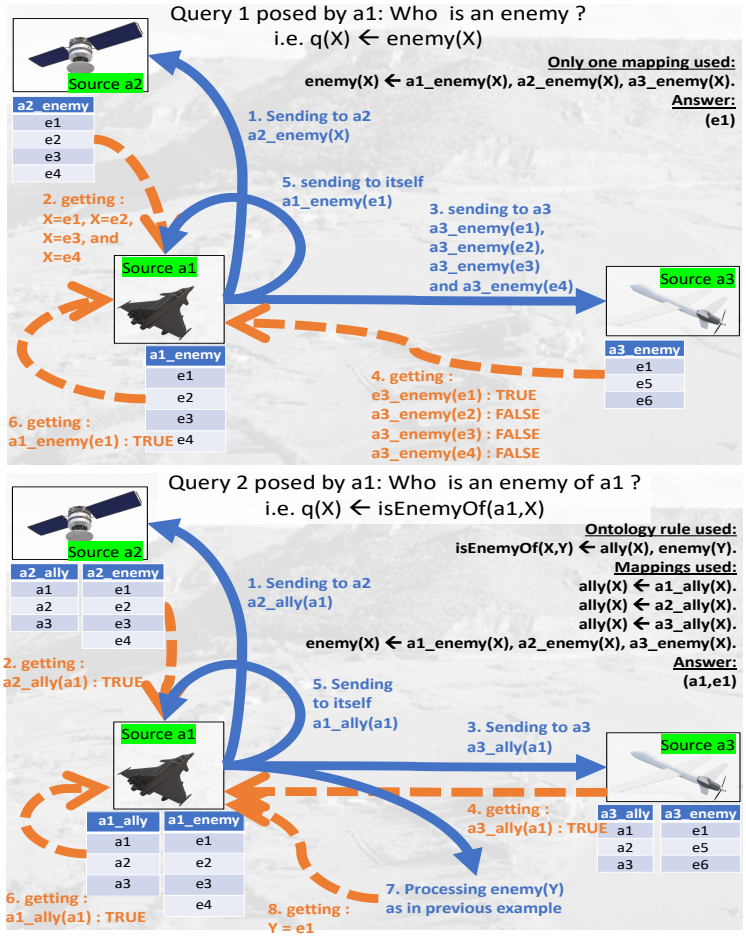

FIG. 3: Example of the rewriting process applied to 2 queries. The rewriting process follows the steps numbered from 1. to 6. for query 1, and from 1. to 8. for query 2.

In the context of e.r., query answering algorithms usually amounts in generating all possible rewritings of the query using the predicates of the data sources schemas and then evaluating these rewritings on these sources. This is a top-down or backward chaining reasoning approach. These rewriting algorithms can be grouped into two categories. First, there are algorithms that reason on decidable restrictions of e.r. They usually are ad-hoc to the supported restriction and still remain largely academic. The second category of algorithms is based on those in the field of l.p. They require translation of e.r. into defined programs or datalog rules, with Skolem functions in order to express existential variables. These algorithms are based on first-order logic resolution (such as SLD-resolution, OLDT-resolution, or SLG-resolution [5]). They are theoreti-

5. These three resolutions are algorithmic schemes that infer logical consequences from knowledge

cally undecidable because of some cases of infinite loops. In practice, however, they can be made decidable, for instance by imposing a maximal execution time. The great benefit with resolution-based approaches is that they are well-known and optimized because they have been industrialized for a long time (see all prolog implementations for example). Thus, we have chosen l.p. as our algorithmic approach. This means that after modeling the ontology and mappings in OWL2, we translate them into e.r., and then translate e.r. into l.p. rules with Skolem functions (a.k.a. datalog rules with Skolem functions). For instance, function $f1$ used in the rules R1 and R2 given in section 3.1 is a Skolem function. The reasoner we have chosen that deals with such rules is the so-called JENA backward chaining rule engine, which belongs to the JENA software [6] and implements a top-down rewriting algorithm based on SLG-resolution. JENA has also been chosen since it is one of the most popular open source industrial platforms [7] to store and query RDF data triples based on the semantic web approach.

Figure 3 shows the rewriting process. Query 1 is posed by a1, and rewritten using M1 into a rewriting that is first sent to a2 which generates 4 possible values for X. These values are sent as 4 rewritings to a3, and only one is ok for a3. Since it is ok for a1 too, then it generates one solution which is e1. Query 2 is also posed by a1 and rewritten using R3, M1, M3, M5 and M7. Steps 1. and 2. involve actor a2, steps 3 and 4 involve a3, and steps 5. and 6. involve a1 itself. Last steps, 7. and 8., need the processing of a rewriting on the same way as query 1 (in 6 steps). These examples show the rewriting approach doesn't need to copy all data from all actors in one place, but instead sends rewritings. This fits well with the context of air missions.

4 Performance Analysis

We have made first experiments on our prototype using a scenario provided by Dassault Aviation, the SITAC ontology augmented with specific knowledge and mappings, and a set of 7 queries. Up to now, we have only been able to test the non-distributed case in which all data are directly accessible from all actors. It does not change the way queries are processed (by rewriting them). It only lacks the communication times and interruptions the distributed architecture may imply. Results are shown in figure 4. First, we have measured execution times for the different steps of query processing: (1) resource creation in memory (initialization of data structures), (2) reasoner object creation in memory, (3) assertions (data) loading from the triple store, (4) reasoner setting, (5) inference time (inference of all query answers, in a backward chaining fashion), and (6) displaying of results. Second, we have measured the number of basic operations during inference, namely extensional (i.e. data source) predicate accesses and rule applications (i.e. basic rewriting steps).

Figure 4 shows that executing the JENA engine on the scenario takes less than 1 sec., which fits with the real time constraint. We also see that inference (query processing) time is around 0.1 sec. The most expensive step is loading the data into

expressed as first-order logic programs. OLDT- and SLG-Resolutions are SLD-Resolution optimizations. For example, SLG-resolution is the basis of the XSB prolog `http://xsb.sourceforge.net/`.

6. See `https://jena.apache.org/`.
7. See the ranking at `https://db-engines.com/en/ranking/rdf+store`.

memory. In a real implementation, this step may be achieved once at the beginning of the mission for data that will not change during the mission. Thus, the data update mechanism might not be as time-consuming since focused on fewer data.

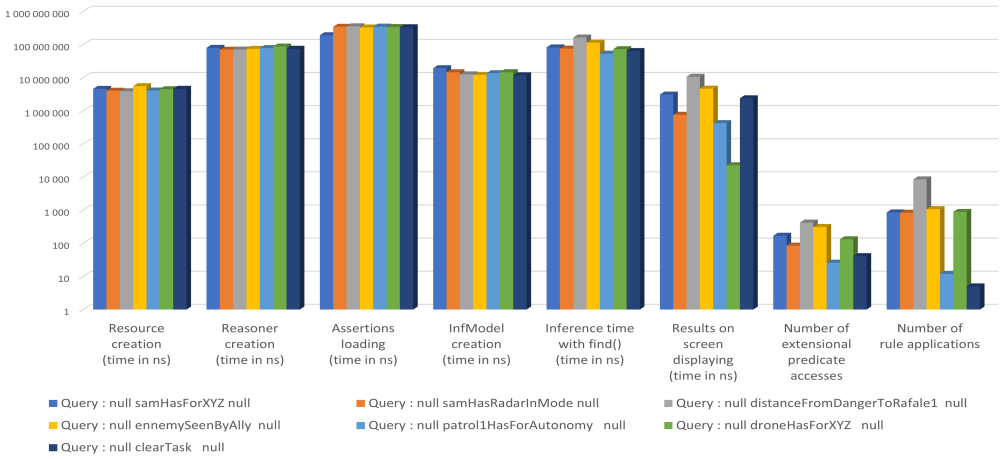

FIG. 4: Experimentation results based on the JENA backward chaining rule engine.

5 Related works

As in this work, [3, 1] work on the same idea of distributed mediation which amounts to sending subqueries (or rewritings) to others actors in order to obtain extra final answers. However, they ground their rewriting mechanism on the Query-SubQuery (QSQ) algorithm [11], whereas we use SLG-resolution [10]. As QSQ, SLG-resolution ensures termination when no function are used, but it can also be used with functions (e.g. Skolem functions) which allows us to use it on translated e.r. QSQNet is a recent extension of QSQ [6] which also handles functions. We have not chosen to use it because its implementation seems not mature enough yet (e.g. it lacks some built-in functions). However, as evoked in [8], no comparative study between QSQ approaches and SLG-resolution has been proposed so far.

6 Conclusion

We have proposed a mediator crowd architecture that can be used in a military air mission context. We have justified the use of an OWL2 domain ontology with RDF data triples and of a top-down rewriting query process relying on the datalog with Skolem functions rule language. We have shown the JENA rule engine could be used to implement the whole approach. Yet, some issues remain open, as robustness (how to deal with communication interruptions?), expressivity (how to add negation or disjunction in the ontology?), and semantic caching (how to optimize performances by caching often used queries and the associated rewritings?).

References

[1] S. Abiteboul, Z. Abrams, S. Haar, and T. Milo. Diagnosis of asynchronous discrete event systems: Datalog to the rescue! In *ACM PODS*, 2005.

[2] S. Abiteboul, I. Manolescu, P. Rigaux, M.-C. Rousset, and P. Senellart. *Web Data Management*. Cambridge University Press, USA, 2011.

[3] M. Al-Bakri, M. Atencia, S. Lalande, and M.-C. Rousset. Inferring same-as facts from linked data: An iterative import-by-query approach. In *AAAI*, 2015.

[4] M. Bienvenu, M. Leclère, M.-L. Mugnier, and M.-C. Rousset. Reasoning with ontologies. In Pierre Marquis, Odile Papini, and Henri Prade, editors, *A Guided Tour of Artificial Intelligence Research, vol. I.* International Publishing, 2020.

[5] A. Calì, G. Gottlob, T. Lukasiewicz, and A. Pieris. Datalog±: A family of languages for ontology querying. In *Datalog Reloaded*, 2011.

[6] S. T. Cao and L. A. Nguyen. Query-subquery nets for horn knowledge bases in first-order logic. *J. Information Telecommunication*, 1(1), 2017.

[7] G. Gottlob, G. Orsi, and A. Pieris. Ontological query answering via rewriting. In *Proceedings of ADBIS '11, Vienna, Austria*, 2011.

[8] M. Kifer and Y. A. Liu, editors. *Declarative Logic Programming: Theory, Systems, and Applications*, volume 20. ACM and Morgan and Claypool, 2018.

[9] M.-L. Mugnier and M. Thomazo. An introduction to ontology-based query answering with existential rules. In *Reasoning Web Summer School, Greece*, 2014.

[10] T. Swift and D. Warren. Xsb: Extending prolog with tabled logic programming. *Theory and Practice of Logic Programming*, 12, 12 2010.

[11] L. Vieille. Recursive axioms in deductive databases: The query/subquery approach. In *Expert Database Conf.* Benjamin/Cummings, 1986.

Acknowledgements

We thank the D.G.A. of the French government for funding this work, and the Dassault Aviation company for managing the IBC project and also for its support.

Résumé

Le projet IBC (Intégration de Bases de Connaissances) s'intéresse à un problème d'intégration de données via une ontologie. Il vise à combiner des données stockées chez différents acteurs (avion, drone, satellite, ...) durant une mission aérienne et à donner aux utilisateurs une vue unifiée de toutes ces données dans un environnement de communication contraint. Nous décrivons la solution que nous avons implémentée, basée sur la médiation. On utilise un langage à base de règles pour traiter les requêtes à partir d'une ontologie du domaine modélisée en OWL2 et de données stockées sous forme de triplets RDF. Nous donnons aussi une analyse des performances du prototype.