A Multiagent Framework for Querying Distributed Digital Collections

Jan de Mooij, Can Kurtan, Jurian Baas and Mehdi Dastani
Intelligent Systems, Department of Information and Computing Sciences, Utrecht University, The Netherlands
{*a.j.demooij, a.c.kurtan, j.baas, m.m.dastani*}*@uu.nl*

Keywords: Multiagent System, Information Retrieval, Federated Search, Semantic Web.

Abstract: Since initial digitization strategies are often inspired by existing usage of the data, and usage of archives often varies among institutes, there is a lot of variation in accessibility of digital collections. We identify four challenges that researchers may encounter when querying such collections in conjunction for research purposes, namely *query formulation*, *alignment*, *source selection* and *lack of transparency*. We present a multiagent architecture to help overcome these challenges and discuss an prototype implementation of this framework. By means of a query scenario we show the utility of using the framework for humanities researchers.

1 INTRODUCTION

Institutions start to realize that it is not just the size of their data collections that determines their usefulness, but also their accessibility. Since initial digitization strategies are often inspired by existing usage of the data, and usage of archives often varies among institutes, there is a lot of variation in accessibility of digital collections. Moreover, the distributed nature of data sources has resulted in using various standards among these sources. Although Digital Heritage and Semantic Web aim at making it possible to use data from multiple data providers, there are still many challenges for organizing and integrating these digital collections.

We aim at enabling integrated search on data collections while keeping them distributed and without forcing changes to the source of the data. Moreover, we aim at enabling non-technical users to search distributed data collections. In order to achieve these objectives, we focus on the following challenges.

1. **Query Formulation:** The users of the system should be supported in formulating queries as they cannot be assumed to be able to formulate the correct query right away. Users may be non-technical, not know exactly what to look for, or realize a query can be improved upon encountering new data.
2. **Alignment:** Data collections may use different schemes to represent entities even for overlapping content in the same domain. This problem may appear at both the data records and schema representation of the collection. For a large number of collections it is not feasible to solve these interoperability problems manually, and therefore semi-automatic alignment approaches are required.
3. **Source Selection:** Complex queries that need data from various collections require careful selection of data sources since the quality of answers are highly related to the relevancy of participating sources. This requires automatically deciding and querying relevant sources, and combining the resulting data. These require in turn some meta-knowledge about the data sources.
4. **Lack of Transparency:** The above mentioned automated processes may affect transparency as users may no longer understand how the answer was formed. A user needs to understand what decisions were made by the system and for what reason.

We focus on the data collections of digital heritage, where the Resource Description Framework (RDF) has been widely used. We assume that digital collections can be queried using the SPARQL query language and that they use different ontologies, or use different identifiers for the same entities. We propose a multiagent system to meet these challenges in a structured manner. The system consists of three agent types organized in a hierarchical structure. The *user agent* supports a user to formulate queries and get results back. The *broker agent* selects data sources, distributes sub-queries, and aggregates the results. Finally, the *data source agent* encapsulates a digital collection to provide data for a request coming from the broker. A query scenario is used to evaluate and illustrate the working of the prototype multiagent system.

2 A MULTIAGENT SYSTEM ARCHITECTURE

In this section we describe the proposed role of the user, broker, and data source agents in an architecture for a multiagent system. The hierarchy of these agents is presented in Fig. 1. We assume a *general ontology* that fully describes the relevant domain in a consistent and unambiguous manner is present. Participating collections are aligned with this ontology, rather than pairwise with each other.

The query process starts with a user who is interested in a question relating to facts in one or more connected data collections. This user interacts with the system through the user agent, which helps the user express the question formally and constrain or loosen it where necessary. The query is then passed on by the user agent to a broker agent. After the broker has collected the answers, the results are presented back to the user by the user agent. At this stage, the user can download the results for further analysis, or work together with the user agent to further improve the query based on the provided results. The main purpose of the user agent is to hide the complexity of formal query writing for distributed data collections from novice users.

The main task of the broker agent is to organize the query distribution process and automated data integration. The broker agent uses the general ontology to learn which concepts exist across the participating collections, and contacts individual data source agents to learn which specific concepts occur in their respective collections, represented by their *capabilities*. The broker agent uses these capabilities to select sources which can provide partial results to incoming queries. When a new query is received, the broker agent splits it into one or more subqueries, which can each be answered by at least one source agent, according to the stored capabilities. These agents are asked to provide the answers for their part of the user query, and the broker agent joins these intermediate results to form the final answer to the original query. Although the user agent itself will contact only one broker, multiple broker agents which can work together to answer a given query may exist in the system.

Each digital collection is managed by a data source agent that can be understood as an expert on the data it encapsulates. Source agents are capable of querying SPARQL endpoints, making them work with existing RDF data structures, or of exposing raw database files. The main task of the source agent is to answer specific subqueries sent to it by a broker agent. The broker agent will be agnostic about the ontology used in the local collection, so the source agent translates the subquery to that ontology, and makes sure results that are sent back are presented in the vocabulary of the general ontology. In order to be included in the query process, a data source agent constructs a local expertise model, representing the capabilities of the collection it encapsulates, and shares that expertise with the broker when requested. The model is constructed by indexing concepts used in the local collection through automatically generated queries, and communicated in the vocabulary of the general ontology.

Data providers can instantiate source agents for their own collections, and announce that agent to the multiagent system to make their data available. They can use the default agent implementation, or make changes to reflect their own values. For example, fine-grained access control not possible with regular SPARQL endpoints, or domain specific reasoning to improve result quality, could be included. Most importantly, however, this agent can run on a server maintained by the data provider itself, so they remain in full control.

3 PROTOTYPE IMPLEMENTATION

In this section we discuss a first prototype implementation of the architecture proposed above, which we intend to extend to a fully functional application in future work. This prototype has been created using the Java-based edition of the agent programming language 2APL (Dastani, 2008; Dastani and Testerink, 2014). All agent communication, including query requests and the sending of query answers, occurs through message passing. In this implementation, the general ontology we use is one specific to the domain of cultural heritage. For all participating digital collections, a mapping to this ontology is provided in separate files (Zamborlini et al., 2017). Additionally, *linksets* containing equivalence relations between entity IRI's across collections are provided (Idrissou et al., 2018). By means of the general ontology and linksets, the broker and data source agents align the data at both ontology and instance levels during query processing.

Whenever the agents need to perform RDF or SPARQL related operations, Apache Jena[1] is used. Permitted queries are restricted to the set of non-nested conjunctive SPARQL queries with a few operators only. This means each query can only be a conjunction of triple patterns (SUBJECT-PREDICATE-

[1] https://jena.apache.org/

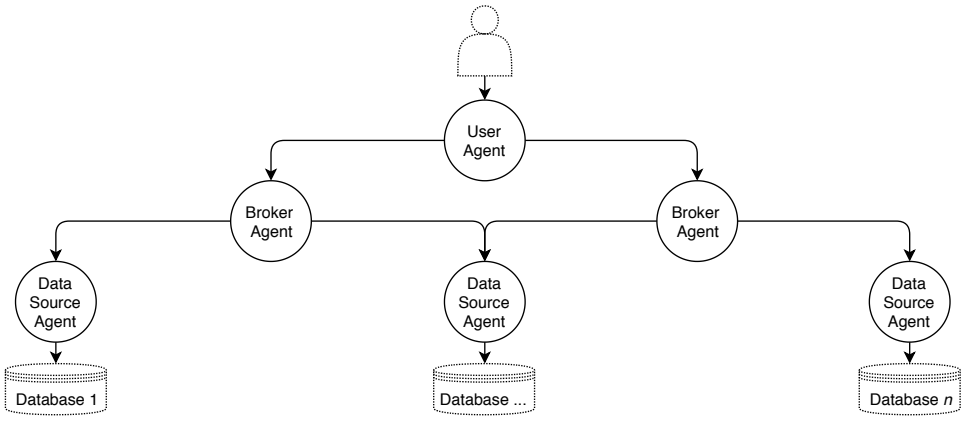

Figure 1: An overview of the multiagent system.

OBJECT triples with variables in zero or more locations), FILTERs, and BINDs.

3.1 User Agent

A user interacts with the user agent through a web interface. Expert users are given the opportunity to write formal SPARQL queries, similar to existing SPARQL endpoints such as the Virtuoso interface of DBpedia[2]. This allows expert users to write SPARQL queries over distributed collections without knowledge of availability or content of individual collections. In ongoing work we are investigating methods of making the query writing process more accessible to non-technical users.

When the user is finished with constructing the query, the user agent starts the querying process by performing a syntax check. If any errors in the query syntax are found, suggestions for repairing the query are presented to the user. Otherwise, the broker is asked to start the query process. When the results are ready, these are displayed in a table where the user can inspect the results, or export them as a CSV file for further analysis. For each variable in the query, all data sources that contributed some values are listed, allowing the user to trace answers back to the source. This last feature is a first step towards transparency of the process.

3.2 Broker Agent

The broker agent uses the process given in Algorithm 1 to answer a user query. First, it creates a set of candidate sources for each triple pattern in the query. A source agent is marked as a candidate for a triple pattern if all concepts occurring in that triple also are within the capabilities of that source. From the candidate sources, each triple pattern is matched with at least one data source agent, ensuring the entire query is covered (line 1). Each data source in the final selection receives an abstract representation of the triple patterns matched to that data source, and is expected to reply with a collection of RDF triple patterns (line 3). The broker agent collects all these triple patterns in a Jena model, to create a temporary virtual graph containing the aggregated data (lines 5-8). The broker adds the linksets of all participating data sources to this model (line 10), and uses the Jena reasoner to materialize the equivalence relations (line 11). This ensures that when different identifiers are used across collections, they resolve to the same entity. Finally, the broker evaluates the original SPARQL query on the temporary model (line 12), and streams the result back to the user agent.

Algorithm 1: *answerQuery*$(Q, A, L, reasoner)$.

Data: Query Q, Set of Agents A, Linkset L, Reasoner *reasoner*
Result: Resultset R

1 $SQ \leftarrow \mathsf{split}(Q, A)$ // create subqueries
2 **foreach** $subQ \in SQ$ **do**
3 | $\mathsf{send}(subQ)$ // send to agent
4 **end**
5 $M \leftarrow \mathsf{init}()$ // initialize model
6 /* async partial data collection */
7 **foreach** $P \leftarrow \mathsf{collect}()$ **do**
8 | $M.\mathsf{add}(P)$ // add partial model
9 **end**
10 $M.\mathsf{add}(L)$ // add linkset
11 $M.\mathsf{reason}(reasoner)$ // reason over data
12 $R \leftarrow M.\mathsf{execute}(Q)$ // execute query
13 **return** R

[2]https://dbpedia.org/sparql

3.3 Data Source Agent

When a data source agent receives a query request, it generates a `CONSTRUCT` query that uses all the triples specified by the broker. With the `CONSTRUCT` keyword, SPARQL allows aggregating results as an RDF graph, which is a temporary virtual database constructed based on the query. By using concepts of the local collection in the query body, but taking the translations of those concepts from the general ontology and placing them in the head, results are automatically returned to the broker agent in the vocabulary of the general ontology. More complex mappings can be dealt with this way as well. An example of this approach is provided in Listing 1, which assumes the general ontology describes the class `schema:Person` and a property of that class `schema:name`, while the local ontology contains the semantically equivalent property `local:name` but does not contain a class representing People (equivalent to `schema:Person`). However, the local ontology contains a class containing *men* and a class containing *women*. The mapping specified that those two classes together are equivalent to `schema:Person`, so the translation can still occur.

```
CONSTRUCT {
  ?x rdf:Type schema:Person .
  ?x schema:name ?name
}
WHERE {
  { ?x rdf:type local:Man }
  UNION
  { ?x rdf:type local:Woman } .
  ?x local:personName ?name
}
```

Listing 1: Example `CONSTRUCT` query used for translation.

A data source can be created either by specifying the URL of a SPARQL endpoint, allowing existing infrastructure of knowledge institutes to be used, or by giving it direct access to the source files of the collection formatted as, for example, Turtle, TDB(2), N-Triples or JSON-LD. The data source agent autonomously determines query strategies based on the method with which the data is provided. For example, an edge case of SPARQL endpoints that the data source agent deals with is that some of these servers limit the number of results that a single query can yield. If the result size is larger than this number, the SPARQL endpoint will silently drop results beyond that limit without notice. Since the limit is usually constant for a given endpoint, the data source agent can determine if such a limit exists and, if so, what that limit is in advance. When querying the endpoint, it iteratively requests batches of data smaller than this maximum result size by including a `LIMIT` clause, and keeps track of the progress to continue querying until all results are collected.

3.4 Agent Preparation

Since this infrastructure allows data providers to add data source agents to the system at any time, some organization is needed. Besides the main processes described above, each agent type has a specific *initialization phase*, where they prepare to take part in the query process. In this phase, source agents build the expertise model of the digital collection they represent and try to find the query result limit. Broker agents search for and contact available data source agents to request their expertise, in order to get a global idea of knowledge present in the system. User agents prepare by finding the broker agents in the system they can send their user's queries to.

In order to allow agents to find each other in the open multiagent system, we add a directory facilitator. This is an agent that acts as a yellow pages service to all other agents, by matching offered services (e.g. '*answer queries*') to other agents' requirements. For an agent to announce itself to the system, it only needs to contact this directory facilitator, which then informs all relevant agents of the new arrival. This is a common approach in open multiagent systems (O'Brien and Nicol, 1998; Dale and Mamdani, 2001).

4 EVALUATION

We show the utility of the multiagent architecture to humanities researchers through an example query scenario. A digital heritage researcher, Alice, has a hypothesis that with the advancement in book pressing technology, publishing books became easier. She expects to see a trend in historical data where first publications of authors come at an increasingly younger age. She searches information online, but realizes that she has to match published books to authors manually. She then finds three relevant sources exposed through SPARQL, two of which provide personnel data about authors – Ecartico[3] and OnStage[4] – and the other a library archive listing details of those authors' published works – STCN[5]. She writes one SPARQL

[3] http://www.vondel.humanities.uva.nl/ecartico/

[4] http://www.vondel.humanities.uva.nl/onstage/

[5] http://www.stcn.nl

query for each collection, tailored to the data it can provide, but notices that each entity has a different identifier in each collection. Even though there is a file that maps those identifiers, she still needs to match the data manually.

She then learns about the multiagent system in which the three collections are already connected. Alice opens the web page through which she can interact with the user agent in her browser, and writes the query given in Listing 2 to requests all authors and their published works, and uses birth and death dates to calculate what age they were when they first had a work published. Note that prefixes have been omitted.

```
SELECT DISTINCT ?name ?birth
(min(?pubYear) AS ?firstPub)
(?firstPub - ?birth) AS ?age)
WHERE {
?person  a             :CreativeAgent .
?person  :hasName              ?name .
?person  :birthYear           ?birth .
?person  :deathYear           ?death .
?person  :authorOf             ?work .
?work    a                    :Story .
?work    :publishedAs            ?pub .
?pub     a               :CreativeAct .
?pub     :yearPublished   ?pubYear .
}
GROUP BY ?person ?year ?death
HAVING (?firstPub < ?death)
```

Listing 2: A SPARQL query which finds the age of authors when they first had a work published.

The system automatically distributes the query among relevant data sources and collects the results without requiring further input from Alice. The user agent then presents the answers back to Alice. The first 5 results for the query in Listing 2 that were obtained using the prototype implementation are displayed in Table 1. The entire query evaluation process takes roughly 20 seconds in this prototype version for the given query.

Table 1: The first 5 results of the query in Listing 2.

?name	?birth	?firstPub	?age
Vondel	1587	1638	51
Coster	1579	1638	59
Brandt	1626	1644	18
Lescailje	1649	1685	36
Cats	1577	1656	79

Alice exports the results to CSV and uses tools she is already familiar with for the analysis. When she is ready to publish her findings, she looks at the list of consulted collections provided by the user agent so she knows which sources to cite. This scenario shows that the proposed multiagent system can help a researcher find results for a query which requires information from multiple data sources.

5 RELATED WORK

The MUSEUMFINLAND project (Hyvönen et al., 2005) aimed at making data from multiple museums available to non-technical users in an integrated manner. They have developed a general ontology describing the common domain of the museums, and a faceted search browser to provide easy access to the collections. Where our approach aims at facilitating researchers, their approach was primarily aimed at facilitating museum visitors. Moreover, where our approach integrates data *virtually* without requiring changes in the collections themselves, in the MUSEUMFINLAND project museums were asked to align their data to the general ontology and uploaded their data to a central repository.

Since RETSINA (Sycara et al., 1996) has been proposed, the hierarchical multiagent structure with user, broker and data source agents has been a common approach to access information sources. The authors describe how process models can be used by agent-based semantic web services to indicate what inputs are required for the services they provide (Sycara et al., 2004). Broker agents reason over these required inputs and either use other services to acquire missing information, or request that information from the user directly. While we use a similar organizational structure, our contribution can be distinguished in that it deals with the problems users may face when querying distributed heterogeneous collections.

Garcia *et. al.* (García-Sánchez et al., 2008) have noted the variation in standards in semantic web and compared those to relatively rigid web standards and protocols. They have proposed a multiagent system that uses the stricter web standards to discover web services that publish semantic data and use their service descriptions to map those data to a domain ontology automatically. A separate ontology is used to store collections' capabilities, allowing agents to jointly reason over discovered services. This approach was aimed at the automated integration of semantic data and to minimize the amount of human intervention required, rather than at accessible search. Although the application has been developed for the domain of bioinformatics, the authors claim it could be applied to open environments.

In the Semantic Web literature, various solutions

to distributed querying have been proposed. By far the easiest solution is to consolidate all data in a central data warehouse, as this allows the use of existing query technologies. However, this requires a huge maintenance effort and is only feasible if the data rarely needs updating, and a single entity has full permission to host the data.

With the introduction of the SERVICE keyword in SPARQL 1.1, the W3C formalized a possible solution for querying distributed data (Hartig et al., 2017; Prud'hommeaux and Aranda, 2013). The method allows users to specify different remote data sources for specific parts of a query. This approach expects users to know which remote sources to consult for every piece of data.

A different approach is that of *Federation Engines*. Most research in federation engines focuses on evaluating queries as fast as possible, while maintaining soundness and completeness of results. Various methods have been suggested which can broadly be classified in two categories: limiting communication overhead and efficient joining. Since queries and results are transmitted over a network, decreasing the amount and size of these transmissions can significantly improve communication speed and thus query execution time. Limited communication overhead can be achieved by intelligently selecting only relevant sources, and constraining the query so only relevant results are returned. Engines such as Splendid (Görlitz and Staab, 2011b) and DARQ (Quilitz and Leser, 2008) achieve this by statistically indexing participating sources, while for example FedX uses query-time heuristics (Schwarte et al., 2011). Methods such as Hibiscus (Saleem and Ngomo, 2014) and those proposed by Harth *et. al.* (Harth et al., 2010) use even more advanced data structures. Examples of efficient joining include early join (to drop irrelevant results before these are sent back to the federation engine) and *Nested Join Loops*, where unbound variables in a sub-query are bound with results from previous sub-queries (Görlitz and Staab, 2011b; Schwarte et al., 2011). Görlitz and Staab (Görlitz and Staab, 2011a) provide an excellent overview of the challenges and approaches in federated querying. Saleem *et. al.* (Saleem et al., 2018) have investigated the efficiency of various federation engines over multiple endpoints.

In this work we have not focused on optimization. However, where federation engines require adherence to global standards that cannot be guaranteed in the domain of Digital Heritage, our approach focuses on integrating these types of collections. Moreover, agents are capable to adapt to user intentions as well as imperfect data, and handle distributed data more dynamically. Furthermore, agents can run in a physically distributed fashion, so data providers can control access to their data by maintaining their own agents. Of course, nothing prohibits incorporating existing approaches from federation engines for query optimization, but this has not been the focus of this work.

6 CONCLUSION

In this work we have acknowledged four challenges that researchers may encounter when querying digital collections of various data providers in conjunction: *query formulation*, *alignment*, *source selection*, and *lack of transparency*. To overcome these challenges, we have proposed a multiagent architecture where data providers can expose their data using custom ontologies and users can search these data in an integrated manner. To this end, three types of agents, namely user, broker and data source, collaborate to query the distributed data on behalf of the user. We have implemented a prototype of this architecture in Java and shown its utility through an example scenario.

The limitations of this approach stem from the assumption of the presence of a general ontology, linksets and mappings. While a general ontology is a common requirement for data sources which are not intrinsically aligned, the creation of mappings and linksets between digital collections currently relies on manual labor. We are in the process of developing automated techniques by means of an *embedding*; an abstract representation of entities in a multi-dimensional vector space that encodes a notion of 'sameness' through distance in this space. To achieve this, we are adapting an existing natural language processing approach called GloVe (Pennington et al., 2014) and work based on that approach (Cochez et al., 2017) to the domain of Semantic Web. We intend to have a new type of agent encapsulating such an embedding to create linksets dynamically when new data source agents enter the system or participating sources are updated.

In future work, we further aim to increase assistance to the user by providing more intuitive ways for interacting with the system. Specifically, we intend to provide *guidance* to the user when writing a query by providing suggestions for how a query can be extended. This should make the querying process easier and guarantee the written query will be able to produce answers. At this stage, user studies will be conducted to evaluate usability of the system with novice users. For improved source selection, we

will focus on building a model that summarizes the content of collections and the relations between them more expressively. Lastly, in an open environment, data quality cannot be enforced. Data source agents can have the responsibility of run-time data enhancement when imperfections are encountered. This is expected to reduce the number of crashes on endpoints and increase data availability. Thus, the overall reliability of the system would be improved. We further intend to make agents more adaptable to unforeseen circumstances, such as temporary unavailability of data sources, by incorporating methods suggested by Decker and Sycara (Decker and Sycara, 1997).

ACKNOWLEDGEMENTS

This work was done in the context of the Golden Agents project (www.goldenagents.org), funded by the Netherlands Organization of Science NWO – Large Investments program.

REFERENCES

Cochez, M., Ristoski, P., Ponzetto, S. P., and Paulheim, H. (2017). Global RDF vector space embeddings. In *International Semantic Web Conference*, pages 190–207. Springer.

Dale, J. and Mamdani, E. (2001). Open standards for interoperating agent-based systems. *Software Focus*, 2(1):1–8.

Dastani, M. (2008). 2APL: a practical agent programming language. *Autonomous agents and multi-agent systems*, 16(3):214–248.

Dastani, M. and Testerink, B. (2014). From multi-agent programming to object oriented design patterns. In *International Workshop on Engineering Multi-Agent Systems*, pages 204–226. Springer.

Decker, K. S. and Sycara, K. (1997). Intelligent adaptive information agents. *Journal of Intelligent Information Systems*, 9(3):239–260.

García-Sánchez, F., Fernández-Breis, J. T., Valencia-García, R., Gómez, J. M., and Martínez-Béjar, R. (2008). Combining semantic web technologies with multi-agent systems for integrated access to biological resources. *Journal of Biomedical Informatics*, 41(5):848–859.

Görlitz, O. and Staab, S. (2011a). Federated data management and query optimization for linked open data. In *New Directions in Web Data Management 1*, pages 109–137. Springer.

Görlitz, O. and Staab, S. (2011b). Splendid: SPARQL endpoint federation exploiting void descriptions. In *Proceedings of the Second International Conference on Consuming Linked Data-Volume 782*, pages 13–24. CEUR-WS. org.

Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.-U., and Umbrich, J. (2010). Data summaries for on-demand queries over linked data. In *Proceedings of the 19th international conference on World wide web*, pages 411–420. ACM.

Hartig, O., Vidal, M.-E., and Freytag, J.-C. (2017). Federated Semantic Data Management (Dagstuhl Seminar 17262). *Dagstuhl Reports*, 7(6):135–167.

Hyvönen, E., Mäkelä, E., Salminen, M., Valo, A., Viljanen, K., Saarela, S., Junnila, M., and Kettula, S. (2005). MuseumFinland—Finnish museums on the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):224–241.

Idrissou, A., Zamborlini, V., Latronico, C., van Harmelen, F., and van den Heuvel, C. (2018). Amsterdamers from the golden age to the information age via lenticular lenses: Short paper.

O'Brien, P. D. and Nicol, R. C. (1998). FIPA—towards a standard for software agents. *BT Technology Journal*, 16(3):51–59.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Prud'hommeaux, E. and Aranda, C. B. (2013). SPARQL 1.1 federated query. W3C recommendation, W3C. http://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/.

Quilitz, B. and Leser, U. (2008). Querying distributed RDF data sources with SPARQL. In *European Semantic Web Conference*, pages 524–538. Springer.

Saleem, M., Khan, Y., Hasnain, A., Ermilov, I., and Ngomo, A.-C. N. (2018). An evaluation of SPARQL federation engines over multiple endpoints. In *The Web Conference*. ACM.

Saleem, M. and Ngomo, A.-C. N. (2014). Hibiscus: Hypergraph-based source selection for SPARQL endpoint federation. In *European Semantic Web Conference*, pages 176–191. Springer.

Schwarte, A., Haase, P., Hose, K., Schenkel, R., and Schmidt, M. (2011). FedX: Optimization techniques for federated query processing on linked data. In *International Semantic Web Conference*, pages 601–616. Springer.

Sycara, K., Pannu, A., Willamson, M., Zeng, D., and Decker, K. (1996). Distributed intelligent agents. *IEEE expert*, 11(6):36–46.

Sycara, K., Paolucci, M., Soudry, J., and Srinivasan, N. (2004). Dynamic discovery and coordination of agent-based semantic web services. *IEEE Internet computing*, 8(3):66–73.

Zamborlini, V., Betti, A., and van den Heuvel, C. (2017). Toward a core conceptual model for (im) material cultural heritage in the golden agents project.