Unified Access to Heterogeneous Data Sources Using an Ontology

Daniel Mercier[1(✉)], Hyunmin Cheong[1], and Chaitanya Tapaswi[2]

[1] Autodesk Research, Toronto, ON, Canada
{daniel.mercier,hyunmin.cheong}@autodesk.com
[2] Autodesk, Pier 9, San Francisco, CA, USA
http://www.autodeskresearch.com/

Abstract. The rise of cloud computing started a transition for software applications from local to remote infrastructures. This migration created an opportunity to aggregate and consolidate analogous data content. However, this data content usually come with very different data structures and data terminologies and is usually tightly coupled to one or more applications. With these disparities and restrictions, the analogous data ends up both centrally stored but spread over several disconnected heterogeneous data sources. In this article, we present an approach to aggregate data sources using live data consolidation. The approach preserves the original data sources; and by doing so, prevents associated applications from having to migrate to a new data source. The approach uses an ontology at its core to serve as a common semantic ground between data sources and leverage its stored knowledge to expand query capabilities.

Keywords: Ontology · Databases · Aggregation · Consolidation
Standardization · Query expansion · Materials

1 Introduction

The development of Cloud technologies, remote computing infrastructures, has recently seen a steep growth due to a combination of global network coverage and higher communication speeds. Software companies are progressively moving their applications from being historically deployed locally on desktops to being deployed on these remote infrastructures. This migration provides an opportunity to aggregate data sources. The receiving data warehouses offer extreme storage capacities, resilience from redundancies, traceability over changes and connection to advanced processing pipelines. This newly aggregated data content often holds compatible and analogous data prone to consolidation. The result from consolidation is an increased availability of data on specific topics and richer data diversity.

The aggregation and consolidation of existing data sources is naturally a challenge because the sources are often historically designed, assembled, and

R. Ichise et al. (Eds.): JIST 2018, LNCS 11341, pp. 104–118, 2018.
https://doi.org/10.1007/978-3-030-04284-4_8

optimized for one or more specific applications; and therefore, deeply linked to these applications. A successful aggregation and consolidation of data sources would have to minimize the impact on the consuming applications and their active operations as well as avoid creating copies that could lead to synchronization issues.

Our proposed solution is to use *live data consolidation*; a method where the data aggregation happens at the time of content query to maintain the original data sources. Its core operations consist in the alignment of the data attributes and the homogenization of data structures. The unique aspect of the proposed solution is the introduction of an ontology at the core to serve as a unifying language. The architecture calls for a registry of data sources and an ontology to:

- Store the information to query each data source attribute,
- Serve as a common language to correlate attributes between data sources,
- Expand query capabilities by developing the ontology to new domains of knowledge.

In this article, we address the various considerations to generate such a solution and develop a full-service architecture. The following section goes through a background on known data aggregation and consolidation techniques; and develop the reasons for the choice of technique. Section 3 details the core operations and described a full service architecture. Finally, Sect. 4 illustrates the use of the ontology during query conversion when searching materials data sources.

2 Background on Data Aggregation

There are two primary techniques to coalesce multiple data sources: data integration and live data consolidation. The former combines multiple scattered sources into one larger source. The latter keeps the original scattered sources and only coalesces the data during content retrieval. The main difference stems from the fact that data integration is a one-off operation resulting in a new and larger data source; while live data consolidation adds operations every time a query is received to: First, convert the original query to the query format of the data sources; and second, homogenize the returned data to the requested format. Both techniques use the schema from each data source for content structure, data attributes, data types and other defining information.

2.1 Comparison Between Techniques

Data integration is appealing as a one-off operation but the decision to adopt the technique must also take into account the following considerations: The presence of a larger and more varied data content structure will have an effect on performance. The resulting data source will also likely be built using a data structure composed of a mix of attributes from the original data sources. This

mixed composition will likely affect the consuming applications and require code changes inside the applications to access the new data source. Finally, if the original data sources are built from various database types, the communication protocol might also have to change. A solution to avert these issues would be to keep the original data sources along with the merged data source. But this solution is sensitive to content changes and prone to synchronization issues. In some cases, the management of copies can become far more complex and computationally intense than the initial operation to integrate the data.

On the other end, live data consolidation does not modify the original data sources; but introduces processing operations during each query for:

- Converting the query from its original format to the formats of the different data sources,
- Converting the returned data contents to the requested format,
- Cleaning the data to remove inconsistencies and data overlapping.

When compared to querying a single data source, these repeated operations add an operational time overhead. However, since each data source is independent, these operations can run in parallel. Beyond processing time, the technique is very versatile. It can easily scale up or down with the number of data sources, and it can accommodate a range of query languages and output formats. The technique is not sensitive to content changes but is sensitive to structural changes instead. For this reason, an implementation of this technique must keep track of the format in which the data is stored inside each data source. Finally, live data consolidation stands as a good intermediary solution before data integration. Its history of uses can influence the choice of attributes and data structure during data integration.

The choice between the two techniques comes down to either moving permanently to an integrated source; or keeping the original sources and incurring a processing overhead.

Both techniques have in common the need for *data standardization* to reshape and convert the extracted information into a common form; and *data cleaning* to address the issues of data overlapping and inconsistencies. Data standardization has two components: attribute matching and schema mapping.

2.2 Attribute Matching

Attribute matching is the identification and correlation of attributes between data sources. The attribute matching must consider all variations in attribute naming including homonyms, synonyms, abbreviations, abstractions, and idioms. As illustrated by recent research, effective attribute matching often goes beyond attribute names. Zhang et al. [16] addressed the relation between attributes and data types to filter out interference between data. Liu et al. [8] established a method to compute semantic similarities by considering associated properties. But most importantly, recent advances in natural language processing (NLP) using word embedding introduced robust methods for automated matching [10].

Word embedding has the capability of identifying similarities and analogies beyond lettering equivalences by encoding the semantic relationships in the Euclidean space [6]. The resulting vector-form facilitates clustering of words and idioms. The key to an effective implementation of word embedding is a domain-specific corpus composed of a consistent set of writings on a particular subject to define the initial space. Word embedding has also the capability to support multi-languages but requires for this purpose to have semantic bridges between languages with direct translations to create embedding alignment [5].

Direct attribute matching is a solution to connect two sources. But in order to support a greater number of data sources, it is more efficient to introduce a common semantic ground to bound schema attributes, e.g. Ali et al. [1] introduced the concept of a global schema in a middle-ware service to connect multiple data sources. A more powerful approach is to use an ontology, a Semantic Web technology.

2.3 Semantic Web Technologies

Research in data integration and live data consolidation frequently uses Semantic Web technologies. As envisioned by Tim Bernes-Lee et al. [3], and as formulated by the World Wide Web Consortium: "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries". Noaman et al. [11] introduced an ontology to do data integration from relational databases and assembled the final schema automatically. Zhao et al. [17] created a set of generic rules to automatically convert the data content of any SQL database, directly into the ontology as instances of ontology classes.

On *live data consolidation*, Konstantopoulos et al. [7] took the approach of federating search queries, converting the *SPARQL* queries to the query language of the Cassandra database. SPARQL is the query language for the Resource Description Framework or RDF, a popular format to store ontologies. Liu et al. [8] expanded the approach by combining multiple open-source libraries for the conversion of SPARQL to the query languages of SQL, NO-SQL, Triplestore or XML databases, and by returning the data source contents in RDF format. Ontology-Based Data Access or OBDA, [15], was developed since the mids 2000s to federate relational data sources through an ontology. OBDA focuses specifically on the query aspect. The query language uses SPARQL which is converted to SQL equivalents to retrieved content from the relational data sources. The process uses at its core a mapping between the ontology and the schema of the relational data sources. The query language is bound to the ontology taxonomy and the traversing of the ontology is limited to the capabilities of SPARQL.

3 Proposed Solution

As our data sources are deeply connected to applications, our choice of technique was primarily driven by the impact that an integrated data source would have

on the associated applications. Therefore, our proposed solution is the implementation of a live data consolidation service with an ontology at its core.

3.1 Ontology for Attribute Matching

The use of an ontology, a cross-linked structure of classes and relationships, is a perfect common semantic ground for attribute matching. An ontology offers a semantic rich environment for attributes and has the advantage of situating attributes in their respective contexts. Figure 1 illustrates the basic use of an ontology for attribute matching.

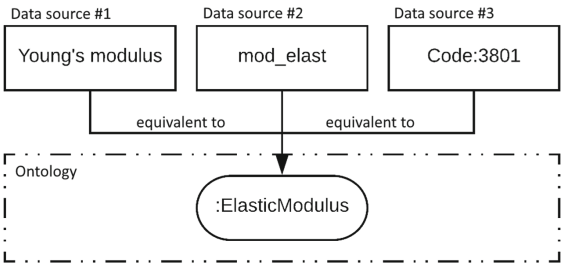

Fig. 1. Attribute matching using ontology

The capacity of an ontology to store extended knowledge adds an important feature to the live data consolidation. Beyond the attribute-bound tree of classes, the exploration of class relationships inside the ontology opens new query capabilities by connecting neighboring classes belonging to different domains of knowledge. Associations that would normally be unavailable, can now be searched with little data overhead and without affecting data sources. As an example, an attribute called 'material_type' may have a valid value called '*Water*'. '*Oxygen*' and '*Hydrogen*' compose '*Water*'. This knowledge can be added inside the ontology with a separate tree of classes for chemical atoms and the setting of two inverse relationships such as *composedOf* and *composes*. With this ontology, the user can now query for materials *composedOf* of '*Oxygen*' and receive in return the data content related to '*Water*'.

3.2 Description of Schema Mapping

From attribute matching, the next step is schema mapping to convert data contents between data sources. Schema mapping is fundamentally a data manipulation. The process takes data content formatted using an input schema and maps it into the format of an output schema. The complexity of the mapping operations essentially depends on the richness of the schemas, e.g. Mecca et al. [9] investigated how the mapping process changes in the presence of a richer conceptual schema.

Schema mapping uses the attribute matching, the data structure from the source schema and various additional structural connections, e.g. the association between values and units for unit conversion. The attribute matching as well as the internal connections are defined using the original schema, and often stored as metadata inside an enhanced version of the schema. Figure 2 illustrates the addition of metadata inside a JSON schema with the attribute '*ontology*' holding the ontology class matching, and the attribute '*unit*' holding the information on unit.

```
{
    "property_attribute":{
        "type": "object",
        "ontology": [":Property_attribute_classname"],
        "properties":{
            "value_attribute": {
                "type": "number",
                "ontology": [":Value_attribute_classname"],
                "unit":{
                    "default": "#unit_name#",
                    "location": "./#unit_attribute#"
                }
            },
            "unit_attribute": {
                "type": "string",
                "ontology": [":Unit_attribute_classname"],
                "unit": "#unit_name#"
            }
        }
    }
```

Fig. 2. Example of enhanced JSON schema

When describing a schema structure, the schema attributes can be split in two groups: *structural attributes* and *value-carrying attributes*. The schema mapping primarily affects structural attributes. We identified three primary transformations illustrated by Fig. 3:

- **Spreading** which flattens trees of attributes.
- **Inverting** which reverses trees of attributes. This transformation creates attribute duplicates with a parent attribute becoming a child attribute to its original child attributes.
- **Condensing** which aggregates branches of attributes into a single attribute.

Condensing was the original drive for associating single attributes to multiple ontology classes. As an example of condensing, let's consider a material with an attribute called '*young_modulus*' matched to the ontology class *:ElasticModulus*.

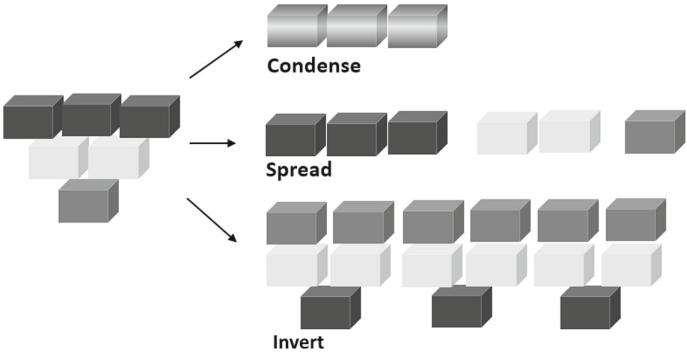

Fig. 3. Structural transformations during schema mapping

This property has two sub-properties: '*value*' attached to the ontology class *:Value*; and '*unit*' attached to the ontology class *:Unit*. The condensed version of the '*young_modulus*' branches becomes two condensed attributes:

- '*young_modulus_value*' attached to the classes: *:ElasticModulus* and *:Value*
- '*young_modulus_unit*' attached to the classes: *:ElasticModulus* and *:Unit*

This one-to-many association maintains an accurate attribute semantic significance, or attribute meaning. This is extremely useful for complex attributes with complex or composite names. For this reason, value-carrying attributes are associated to the list of ontology classes in their tree branch during schema mapping. During the development of the schema mapping engine, the presence of these associations helped identifying a few best practices for attribute matching:

1. The matching of structural attributes to ontology classes should be avoided unless there is a direct dependency with a value-carrying attributes, such as, *young_modulus* and *value*. An exhaustive matching of structural attributes reduces the chance of matching sequences of classes between schemas.
2. The order of ontology classes matched to an attribute does not affect mapping because each class is unique.

3.3 Service Architecture

Our implementation of live data consolidation takes the form of a stateless service with two primary workflows and a dedicated query language. The service acts as an agent between users and data sources as illustrated by Fig. 4.

As usual for a service, authentication and authorization protect the service and enforces access rights through user profiles. The four profiles which may interact with the service, are:

- *User* who queries data content,
- *Schema owner* who provides the necessary information about a data source and sets the mapping between the schema and the ontology,

– *Domain expert* who maintains the core ontology,
– *Administrator* who oversees users and operations.

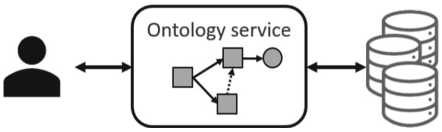

Fig. 4. Simplified view of service.

The set of service operations is tightly linked to these user profiles. While the service comes with a standard set of operations for clustering and user management, the remaining operations cover the two primary workflows for *data source registration* and *querying*.

3.4 Data Source Registration

The workflow for data source registration is designed to gather the necessary information about a new data source from its *schema owner*. The workflow is best supported by a dedicated user interface and facilitate attribute matching. Figure 5 illustrates the workflow. Upon receiving the information about a new data source in the form of its database type, location and original schema, the service immediately returns an enhanced schema with an initial attribute matching. This initial matching is currently automatically generated from lettering equivalence using Levenshtein distance function over the combination of ontology class label and previously matched attributes to that class. In the future, the implementation of word embedding should help implementing a more efficient automatic matching engine. It is worth mentioning that the service does not automatically match attributes to multiple ontology classes as this is left to the schema owner to define richer context for attributes.

During the next phase, the schema owner validates and modifies the attribute matching. To assist with the matching process, the service provides upon request the five closest alternatives to a given attribute. If the schema owner does not find an adequate match for an attribute, the service can provide the taxonomy of the ontology, the bare tree of ontology classes, with a few selected properties to aid in finding a suitable class match. If the schema owner still cannot find a match for an attribute, the schema owner may submit a request to add a new class to the ontology. Upon receiving the request, the service places the data source registration on hold, records the request and transfers the request to the domain experts in charge of curating the ontology. The domain experts in turn, may choose to add a new class or make a recommendation to the schema owner. In both cases, a notification is sent back to the schema owner to complete the registration.

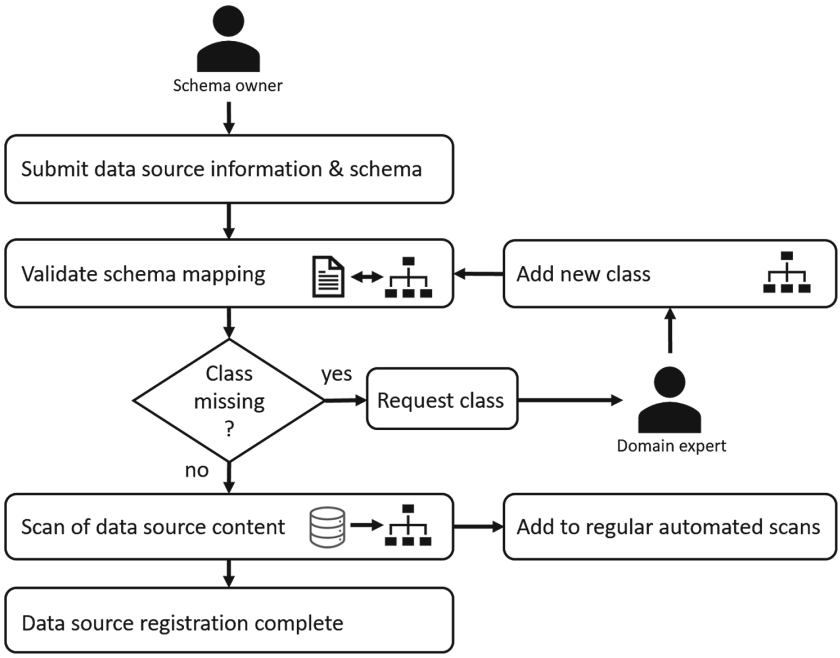

Fig. 5. Sequence during data source registration.

During attribute matching, the schema owner is also responsible for providing a set of '*equivalencies axioms*'. The concept is very important to fully leverage the ontology. Some classes are outside the attribute-bound tree of classes. They are part of neighboring trees holding additional segments of knowledge; but they are not matched to attributes or connected to classes matched to attributes through relationships. These classes may still be linked to schema attributes through equivalency axioms. An equivalency axiom is illustrated by Eq. 1. It links an ontology class to a query expression recognized by the data source. These equivalency axioms are used during query conversion.

$$: Isotropic <=> \text{“}material\_structure\text{”} = 1 \tag{1}$$

Once the service receives the final schema complemented with attribute matching, internal connections, and equivalency axioms, the service can complete the data source registration. The source general information is added to the source registry, the schema is parsed to extract operational data, and the attribute matching is modified to aggregate for each value-carrying attribute, the sequence of classes along its schema branch. Finally, for each listed equivalency axiom and value-carrying attribute, a new *individual*, an instance of an ontology class, is added to the ontology to store the necessary information in the form of individual '*data properties*' to build a query for the newly registered data source.

The number of data properties depends on the type of database, and whether the individual holds information about an attribute or an equivalency axiom. The data properties may include the data type as well to check for type consistency and unit for unit conversion.

The individuals are named after their parent class names. The individuals are grouped under unique IRIs for each data source. This construction allows easy identification of the content stored inside the ontology and facilitates maintenance operations. When an attribute is attached to more than one class, its individual is attached to each of these classes and its name composed by combining class names, alphabetically sorted. The naming convention is designed for easy parsing using regular expression.

During the generation of individuals, the service creates in parallel a two-way dictionary between attributes and classes. This dictionary is used during the initial phase of query conversion from the original query language to a composition based on ontology classes. After the data source registration is complete, the newly added data source joins the pool of already available data sources and becomes available for users to query.

3.5 Query

Users have access to a query function to search the aggregated data content from the registered data sources. Figure 6 illustrates the query workflow. It hinges on the initial selection of a schema by the user. The schema sets the language for the original query and the format for the returned data content.

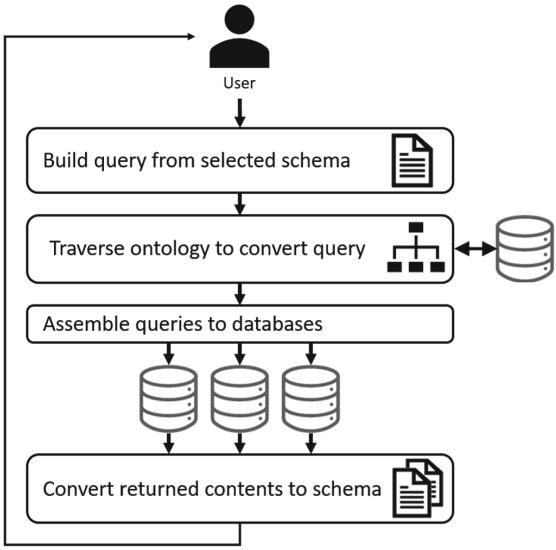

Fig. 6. Sequence during a query.

The query language is a combination of a generic query framework and a well nested declarative environment. The query framework includes logical operators (AND, OR, NOT) as well as parentheses to create groups. The nested declarative environment is generated by merging two dictionaries and adding the list of ontology relationships and associated class names. The first dictionary is the one mentioned in the previous section, extracted from the user schema between attributes and ontology classes. The second dictionary is extracted from the ontology itself with the associations between ontology class labels and ontology class names. The two dictionaries are merged by replacing any equivalent attribute/class entry inside the ontology dictionary by the one found inside the schema dictionary. Once composed, the nested declarative environment is used to do the initial conversion of the query to an intermediary form with ontology class names in place of attributes. The nested declarative environment is also used to implement an auto-complete feature to assist with early query composition and partial validation. The auto-complete feature can suggest attribute names from partially typed attributes, isolate invalid ones, check relationships, validate value types, and verify unit compatibility.

The format of a generic query is a set of logically bounded and grouped functional blocks. The query parser assumes a linguistic typology of "*subject* verb *object*" for the query blocks. The generic subject is assumed to be '*data*'. The '*object*' can be either be a logical expression, a numerical expression, or a string expression.

- A logical expression tests existence, and the '*object*' should be a single term,
- A string expression associates a term and a specific string with equality '=',
- A numerical expression associates a term and a value with a comparison operator =, <>, <, >, <=, >=. If the block includes unit and the unit information was provided for the intended data source, the value is converted, and the unit name removed.

In the presence of a '*verb*' in the form of a known relationship, the query parser expects a logical expression as '*object*'.

After the initial conversion of *objects* from attributes to ontology classes, the query parser accesses the ontology to find the necessary individuals to continue the conversion into the specific query language of each data source. In the case where an individual for a target data source is not directly connected to the class found as the *object*, the query parser starts exploring the ontology to find one.

- If the functional block does not include a relationship as '*verb*', the query parser assumes either an '*is*' as a defining state or a '*has*' as a defining property; and explores from parent to children.
- If the functional block does include a relationship as '*verb*', the query parser first traverses the relationship and then explores from parent to children. If an identical relationship is detected during the exploration, the query parser traverses the new relationship; and from there, continues the exploration.
- If multiple individuals are found during the exploration, the original block is converted into as many blocks as the number of found individuals.

The key to the query conversion is the effective exploration of the ontology to find individuals attached to data sources. The process can partially be driven by the SPARQL query language. However, the SPARQL language is explicit which limits lateral exploration over unknown layers of class relationships. The work of Reuter et al. [13] proposed to solve the issue by adding an additional keyword to the SPARQL language for recursive searches. Our service uses a dedicated programming language for the exploration and traversing of the ontology in place of SPARQL.

The outcome of the query conversion is a set of new queries for each of the registered data sources. If the query parser fails to compose a new query, the associated data source is excluded. Once composed, these new queries are sent to their respective data sources. The returned data content is then converted using schema mapping to the user schema. During this operation, if units are specified in both the source and destination schemas, values as well as unit names may also be converted.

The combination of attribute matching, schema mapping, operational workflows, simplified query language with the use of an ontology at the core creates an effective and functional body for the live data consolidation service.

4 Application

The aggregation of materials data sources is the original motivation for this article. Applications in Computer Aided Design, Computer Aided Manufacturing, and Computer Animation, all rely on materials data to accurately represent and simulate reality in the digital world. However, materials data is historically scattered and often application specific. The intent for building a live data consolidation service for materials data is to facilitate access to these data sources, increase the available content, and by doing so, open new research in data validation, characterization, surrogate modeling, and the discovery of new materials. The core for this implementation of the live data consolidation service is its material ontology.

4.1 Materials Ontology

The history of material ontologies is a mix of generic and dedicated approaches. One of the earliest published materials ontology, the Plinius ontology [14] focused on ceramic materials. The effort was followed by many other ontologies with diverse degrees of refinements. More recently, Premkumar et al. [12] established the Semantic LAminated Composites Knowledge management System or SLACKS, to bridge composite materials and their manufacturing processes. On the generic side, Ashino [2] established one of the earliest set of classes for materials. Cheung et al. [4] created a platform called MatOnto for materials data integration using a generic ontology to facilitate research.

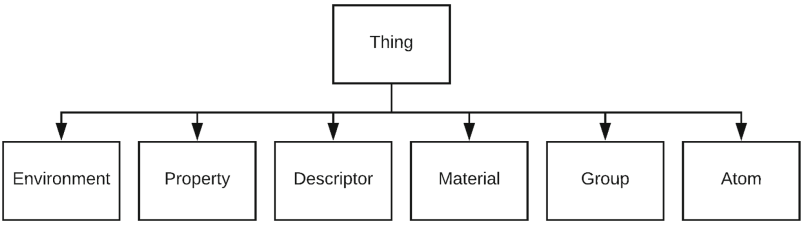

Fig. 7. Top layer of the materials ontology.

Our base ontology is inspired by the work of Ashino [2]. Ashino was one of the first to establish a well-recognized generic ontology structure to describe materials. His ontology focused on material types, families, environment and properties. While Ashino designed his ontology for direct storage of the data inside the ontology, our ontology only serves as a middleman to unify data sources. As such, Ashino's instance properties, *Object Properties* and *Datatype Properties* are data source attributes in our context and present in the ontology as standalone classes instead. The first layer of our ontology is illustrated by Fig. 7. This materials ontology before service activation has an initial set of around 50 classes. The ontology was also enhanced with the periodic table to connect materials to their chemical compositions. The periodic table adds about 120 classes and 200 individuals.

4.2 Query Examples

Here is a list of query examples for materials data.

Use of an equivalency axiom - Using the Eq. 1, and given the query:

$$query: \quad NOT \quad Isotropic \tag{2}$$

'*Isotropic*' is first replaced by the class '*:Isotropic*' and then replaced by the expression '($material\_structure = 1$)' upon finding the equivalency axiom inside the ontology. The results are materials with '*material_structure*' different from 1; which covers mostly composite materials, such as:

- Fiber reinforced polymers from polymer data sources,
- Reinforced concretes from construction data sources.

Traversing the ontology using class relationships - Given the query:

$$query: \quad isNamedAfter \quad Acrylate \tag{3}$$

'*Acrylate*' is converted to the class *:Acrylate* and this class is linked through the '*isNamedAfter*' relationship to four classes. Each with an equivalency axiom linking to the polymer data source:

- :AcryloNitrile with '*category* = acrylonitrile'
- :MethylMethacrylate with '*category* = methylmethacrylate'
- :PolyCyanoAcrylate with '*category* = methylmethacrylate'
- :PolyMethylMethAcrylate with '*category* = polymethylmethacrylate'

The results are the grades (commercial types) for the above categories.

Combined query capabilities - Given the query:

$$\begin{aligned} query: \quad & Polymer \quad \text{AND} \quad young\_modulus > 500 \quad ksi \\ & \text{AND} \quad young\_modulus < 550 \quad ksi \\ & \text{AND} \quad density > 0.2 \quad lb/in^3 \\ & \text{AND} \quad density < 0.5 \quad lb/in^3 \end{aligned} \tag{4}$$

- '*Polymer*' becomes the class '*:Polymer*' and access the first source with the equivalency '*material_type = polymer*' and access the other source by combining children classes with the insert '*family = thermoplastic* AND *family = thermoset* AND *family = elastomer*',
- '*young_modulus*' becomes for one source '*elastic_modulus*' and for the other '*young_mod*',
- One of the source uses SI units. The young modulus is searched in '*ksi*' and the stored unit is in '*GPa*'. Therefore, the values and units are transformed during query conversion to '*GPa*' and the returned data content from that source converted back to '*ksi*'. A similar unit conversion applies to the densities.

The results are various grades (commercial types) of:

- Nylon 12
- Polyether Block amid

5 Conclusions

This article introduced an innovative and viable service architecture to access multiple heterogeneous data sources using live data consolidation. The architecture is portable and has a versatile engine for data standardization. The core of its architecture is its ontology. The ontology acts as a common language to federate data structures and attributes between data sources. The ontology also expands query capabilities with little overhead, well beyond the ones made available by the original data sources. Overall, this article illustrates the capacity for an ontology to serve as a preprocessor to connect computing resources and enhance their capabilities by leveraging its stored knowledge.

References

1. Ali, M.G.: A multidatabase system as 4-tiered client-server distributed heterogeneous database system. Int. J. Comput. Sci. Inf. Secur. **6**(2), 10–14 (2009)
2. Ashino, T.: Materials ontology: an infrastructure for exchanging materials information and knowledge. Data Sci. J. **9**, 54–61 (2010)
3. Berners-lee, T., Hendler, J., Lassila, O.: The semantic web: a new form of web content that is meaningful to computers will unleash a revolution of new possibilities. Sci. Am. **284**(5) (2001)
4. Cheung, K., Drennan, J., Hunter, J.: Towards an ontology for data-driven discovery of new materials. In: Semantic Scientific Knowledge Integration AAAI/SSS Workshop, pp. 9–14. Stanford University, Palo Alto (2008)
5. Duong, L., Kanayama, H., Ma, T., Bird, S., Cohn, T.: Multilingual training of crosslingual word embeddings. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, pp. 894–904 (2017)
6. Jurafsky, D., Martin, J.H.: Speech and language processing. https://web.stanford.edu/~jurafsky/slp3/
7. Konstantopoulos, S., Charalambidis, A., Mouchakis, G., Troumpoukis, A., Jakobitch, J., Karkaletsis, V.: Semantic web technologies and big data infrastructures: SPARQL federated querying of heterogeneous big data stores. In: International Semantic Web Conference (2016)
8. Liu, Z., Calve, A.L., Cretton, F., Glassey, N.: Using semantic web technologies in heterogeneous distributed database system a case study for managing energy data on mobile devices. Int. J. New Comput. Archit. Appl. **4**(2), 56–59 (2014)
9. Mecca, G., Rull, G., Santoro, D., Teniente, E.: Semantic-based mappings. In: Ng, W., Storey, V.C., Trujillo, J.C. (eds.) ER 2013. LNCS, vol. 8217, pp. 255–269. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41924-9_22
10. Muzny, G., Zettlemoyer, L.S.: Automatic idiom identification in wiktionary. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pp. 1417–1421 (2013)
11. Noaman, A., Essia, F., Salah, M.: Web services based integration tool for heterogeneous databases. Int. J. Res. Eng. Sci. **1**(3), 16–26 (2013)
12. Premkumar, V., Krishamurty, S., Wileden, J.C., Grosse, I.R.: A semantic knowledge management system for laminated composites. Adv. Eng. Inform. **28**, 91–101 (2014)
13. Reutter, J.L., Soto, A., Vrgoč, D.: Recursion in SPARQL. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9366, pp. 19–35. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25007-6_2
14. van der Vet, P.E., Speel, P.H., Mars, N.J.: The Plinius ontology of ceramic materials. In: Proceedings of Comparison of Implemented Ontologies Workshop (1994)
15. Xiao, G., et al.: Ontology-based data access: a survey. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2108, pp. 5511–5519 (2018)
16. Zhang, R., Wang, J., Bu, W.: Research on attribute matching method in heterogeneous databases semantic integration. J. Chem. Pharm. Res. **7**(3), 16–26 (2015)
17. Zhao, S., Qian, Q.: Ontology based heterogeneous materials database integration and semantic query. AIP Adv. **7**(10) (2017)