Towards a methodology for lossless data exchange between NoSQL data structures

Ronald Rudnicki, Alexander P. Cox, Brian Donohue, Mark Jensen

Ronald Rudnicki, Alexander P. Cox, Brian Donohue, Mark Jensen, "Towards a methodology for lossless data exchange between NoSQL data structures," Proc. SPIE 10635, Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IX, 106350R (4 May 2018); doi: 10.1117/12.2307717

SPIE.

Event: SPIE Defense + Security, 2018, Orlando, Florida, United States

Towards a methodology for lossless data exchange between NoSQL data structures

Ronald Rudnicki[1] , Alexander P. Cox, Brian Donohue, Mark Jensen
CUBRC, Inc., 4455 Genesee St. Suite 106, Buffalo, NY, USA 14225-1955

ABSTRACT

The variety of data structures used by NoSQL databases (e.g. key-value, document, triple store, graph) is evidence of the variety of ways in which data is used within an enterprise. Data in triple-stores that are aligned to semantically rich ontologies are useful for discovery and all-source analysis while data in key-value stores are useful for massive scale high-speed computing. Within an enterprise that utilizes multiple types of NoSQL databases the exchange of data from one to another, if accomplished, usually comes at the cost of losing some amount of content. A methodology of lossless data exchange that utilizes semantic metadata libraries is described.

Keywords: NoSQL, data integration, ontology

1. INTRODUCTION

Recent trends in data management architectures seem to have made the traditional data warehouse obsolete. At the time of this writing, a search for "Is the data warehouse dead" returns nearly 10,000 results. While these announcements of the demise of the data warehouse may be premature, the advent of Big Data did produce a challenge to the warehouse methodology that continues to be difficult to meet. The data warehouse uses a "schema at write" data model, that is, a model created to organize a subset of enterprise data sources so as to meet the reporting and analysis needs of the majority of users within the enterprise. This model purposefully excludes data that is deemed unimportant to those reporting and analysis needs and when this choice is hardened by the large effort needed to modify the model, the warehouse seems unable to deliver on the promise of Big Data. For in Big Data, the thought is that any data may prove to be valuable to someone or some algorithm at some point in time.

An alternative designed to overcome this limitation of the data warehouse is the data lake. A data lake is one or more centralized repositories into which data is deposited in its native format. In theory a data lake contains all enterprise data sources and would keep such data for all time. No data is excluded as being unimportant and storing data in native format is a transition from the "schema at write" approach of the data warehouse to the "schema at read" approach that is intended to enable analysis that transcends the rigid format of a predefined schema.

Terrizanno [1] describe a data lake in which all data sources are kept in native format as a "raw data lake" and argue that such a design fails to enable the performance of data analysis by everyone in the enterprise. Extracting reports and analysis from numerous sources all in their own parochial formats is too time consuming and often beyond the skill set of many data consumers. Some[2] explain that the reporting needs of the majority of users can be satisfied by creating views into the data lake that extract the needed information and that the discovery of new insights is accomplished by data scientists. We find this value proposition of the data lake hard to discern from that of the data warehouse as views merely replace the warehouse reports and the data scientist always had the skill set to go to data sources outside of the warehouse to seek additional insights.

It seems then that we are left with a design choice of a schema at write design that throttles insight or a schema at read that blocks understanding, neither of which are optimal for realizing the potential of Big Data to reveal great insights from large diverse data sets. In Terrizanno a curated data lake is offered as the solution to this dilemma. A curated data lake is one that is populated with data having been processed through stages of procurement, grooming, provisioning and preserving. The resulting repository more realistically enables all users with access to data analysis across the enterprise.

[1] rudnicki@cubrc.org, phone: 716-204-5208, fax: 1-716-849-6655, website: www.cubrc.org
[2] For example: https://www.blue-granite.com/blog/bid/402596/top-five-differences-between-data-lakes-and-data-warehouses

Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IX,
edited by Michael A. Kolodny, Dietrich M. Wiegmann, Tien Pham, Proc. of SPIE Vol. 10635,
106350R · © 2018 SPIE · CCC code: 0277-786X/18/$18 · doi: 10.1117/12.2307717

While a common semantics is seen as being a main component of a curated data lake it is not explained how to accomplish this without falling into the pitfalls of a schema at write design.

The goal of this paper is to describe a functional design to produce semantic interoperability within the data lake and then leveraging the lake to produce different representations of data such as graph representations through automated translations. The approach depends upon the features found in ontologies exemplified in the Common Core Ontologies (CCO), a suite of modular ontologies based upon Basic Formal Ontology (BFO) [2]. In Section 2 we describe the features of ontologies useful to the production of semantic interoperability within these architectures. In Section 3 we explain how the ontologies can be used as the schema of a NoSQL column family database. In section 4 we explain how that schema can be translated to a graph representation. In Section 5 we discuss future work needed to actualize the functional approach.

2. ONTOLOGY DESIGN AND SEMANTIC INTEROPERABILITY

To solve the dilemma of providing the data lake with an at write schema that facilitates understanding and also allowing any and all data to be included, a method of producing a schema that can keep pace with the expansion of the data lake content is needed. We believe that ontologies can provide the needed schema and to that end we examine the Linked Data and Modular Ontology development methods with the conclusion being that the latter is best suited to development at the required pace.

2.1 Linked Data Ontology Development

The Linked Data[3] initiative has the important goal of utilizing the internet to create a shared and open information space by linking documents and other web resources through the use of URLs. In addition, web resources can be annotated with metadata using technologies such as microformats, microdata and RDFa. The metadata includes the type that a given resource instantiates. These types are selected from any vocabulary or ontology of the web designers choosing. The ontologies used in this process are illustrated by The Linking Open Data cloud diagram below in Figure 3.

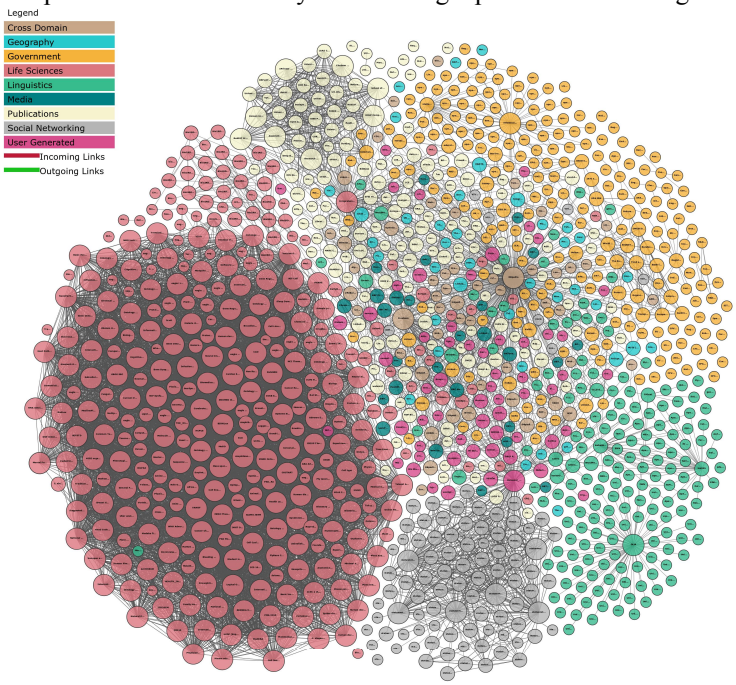

Figure 1 Linking Open Data cloud diagram 2017[4]

[3] http://linkeddata.org/home
[4] by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/

The Linked Data technique for keeping pace with such a vast number of resources is to develop ontologies as needed for a particular domain. The overarching semantics for the entire domain is achieved by creating mappings between classes from different ontologies using the Web Ontology Language (OWL) relation of equivalentClass. This relation carries the meaning that the two classes associated by the relation have the same set of instances. For example the DBpedia Ontology (dbo) class Hospital[5] is asserted to be equivalent to the schema.org (schema) class Hospital[6]. With this axiom in place a web page annotation of Miami's Jackson Memorial Hospital being an instance of dbo:Hospital supports the inference that the hospital is an instance of schema:Hospital also. The benefit of this practice is improved recall of searches since a search across the web for dbo:hospital should return all resources annotated with schema:Hospital too.

The verification of the correctness of all equivalent class mappings is a complex task. Consider that the equivalent class axiom between dbo:Hospital and schema:Hospital made in the DBpedia Ontology leads to a contradiction. In dbo, Hospital is a subclass of Building which is a subclass of Architectural Structure which is a subclass of Place. In dbo, Place is asserted to be disjoint with Agent and Agent is the parent class of Organization. In dbo, Organization is asserted to be equivalent to schema:Organization which is the parent class of schema:MedicalOrganization and which in turn is the parent of schema:Hospital. Figure 4 shows the hierarchies and assertions that lead to this contradiction.

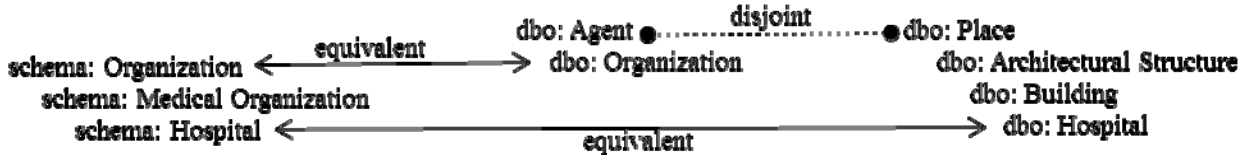

Figure 2 DBPedia and schema.org Hospital Contradiction

In order for the Linked Data approach to succeed in developing a common semantic schema for all data sources in a Big Data enterprise there would need to be both automated methods for developing ontologies of each data source and for establishing equivalent class mappings between terms with common meaning in the disparate ontologies. While applications for creating an ontology for a particular data set are available[7], automated methods for establishing accurate equivalent mappings are not. Basing a mapping on semantic similarity of terms is, as the example above shows, not reliable. Instead, for every pair of terms, the method would have to examine the class hierarchy of both and all axioms of every class in those hierarchies. The number of classes contained in all of the ontologies depicted in Linking Open cloud diagram leads us to the conclusion that this method will struggle to produce a semantically coherent schema for the enterprise in a timely manner.

2.2 Modular Ontology Development

As an alternative to the Linked Data method of developing a schema to keep pace with the velocity, variety and volume of Big Data, we offer a method of modular ontology development. We claim no real novelty in the approach except perhaps in that it leverages the strengths of methods used in Linked Data, the Open Biological and Biomedical Ontology Foundry[8] and object-oriented programming while attempting to avoid their weaknesses.

2.3 Use of an upper-level ontology

Basic Formal Ontology (BFO) is an upper level ontology comprised of 35 classes and 20 properties that depict the world in a highly generalized manner. Like some other upper level ontologies, BFO provides a wireframe view of the world that is intended to be built upon to describe specific domains of interest. A simplified version of the BFO worldview is that qualities differentiate objects and objects participate in events. Events take place over time and both objects and events are located in space. A graphical depiction is provided in Figure 1.

[5] http://dbpedia.org/ontology/Hospital
[6] http://schema.org/Hospital
[7] e.g. http://d2rq.org/generate-mapping
[8] http://www.obofoundry.org/

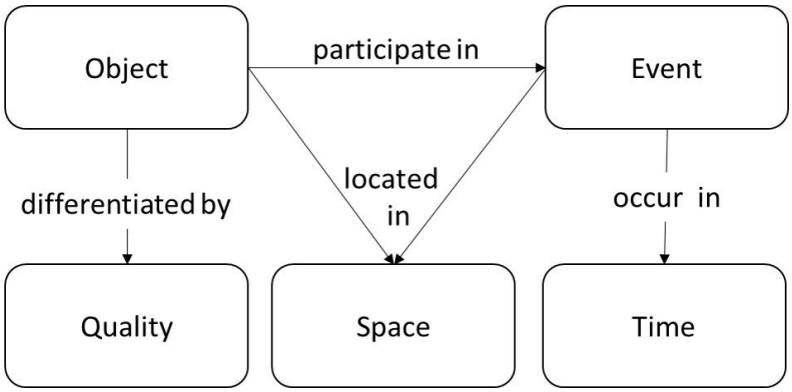

Figure 3 Simplified View of the Basic Formal Ontology

The use of an upper-level ontology carries with it a number of benefits. There is a reduction of effort as well designed upper ontologies contain reasonable treatments of the general relationships between entities thus sparing one the need to re-determine those relationships for oneself. There is improved consistency both within a single ontology and among multiple ontologies that are developed on the same upper-level ontology. If the upper-level ontology specifies the relationship between an object and its qualities, then during the development of a sensor ontology that relationship will form a common pattern between the relations of a sensor system to its capabilities and of the relations of the parts of sensor systems to theirs. The improved consistency carries over to different ontologies based upon the same upper ontology and yield the important programmatic benefits. Queries, algorithms and rules that work on one of the ontologies, should work on the other with little modification. For example the same query that returns all characteristics of an acoustic sensor will return all characteristics of a fixed-wing UAV by changing only the root class.

Reduced time and effort and improved inter and intra-consistency are benefits yielded by using any of the well-designed upper-level ontologies that are available. Using BFO carries additional benefits that made it the choice for the upper-level ontology of the CCO. BFO has a large user base with 50 organizations and 225 ontologies listed on their webpage[9]. This usage rate assures that BFO is well vetted and that by being widely known will aid in the adoption of an ontology based upon it. A user that is acquainted with BFO will be able to understand a BFO-based ontology much faster than one in which they must learn a new upper-level schema. BFO entered the ISO Draft International Standard stage as of December 2017[10]. As BFO becomes an ISO standard its usage rate should increase and it will be afforded with a longevity and stability that reduces the risk of one's adoption of it.

2.4 Extending the upper-level ontology

The development methodology of modularity prescribes the creation of ontologies along two axes: generality and content. Along the axis of generality ontologies are differentiated into upper-level, mid-level and domain-level ontologies. Upper-level ontologies, exemplified by BFO and DOLCE[11] contain classes and relations that are used, if only implicitly, to describe every domain of interest. Mid-level ontologies, of which the Common Core Ontologies (CCO)[12] are examples, contain classes and relations that are sub-types of those contained in an upper-level ontology and which are used to describe many, if not all, domains of interest. Domain-level ontologies contain classes and relations that extend from the content of mid-level ontologies and are used to describe a particular domain of interest. We acknowledge that the distinctions among the levels of generality as drawn here are vague, but in practice they are useful and nothing breaks catastrophically if there are some grey areas between levels.

The second axis, that of content, is determined by the upper level ontology. BFO divides the world into a select set of classes that, open world assumption aside, are the ancestor classes of all the other types of the world. To create the CCO, leaf classes of BFO form the parent of the root class(es) of each of its individual ontologies. Some examples of this are that the BFO class of Three-Dimensional Spatial Region is the parent class of the root class of the CCO Geospatial

[9] http://ifomis.uni-saarland.de/bfo/users
[10] https://www.iso.org/standard/74572.html
[11] http://www.loa.istc.cnr.it/old/DOLCE.html
[12] https://github.com/CommonCoreOntology/CommonCoreOntologies

Ontology, Geospatial Region and the BFO class, Object, is the parent class of the root class of the CCO Artifact Ontology, Artifact. In the same way as the leaf classes of BFO form the starting points of the CCO, the leaf classes of the CCO can form the root classes of domain ontologies as depicted in Figure 2. Dividing content along the axes of generality and content results in a set of disjoint, semantically consistent ontologies with every increasing coverage of the world.

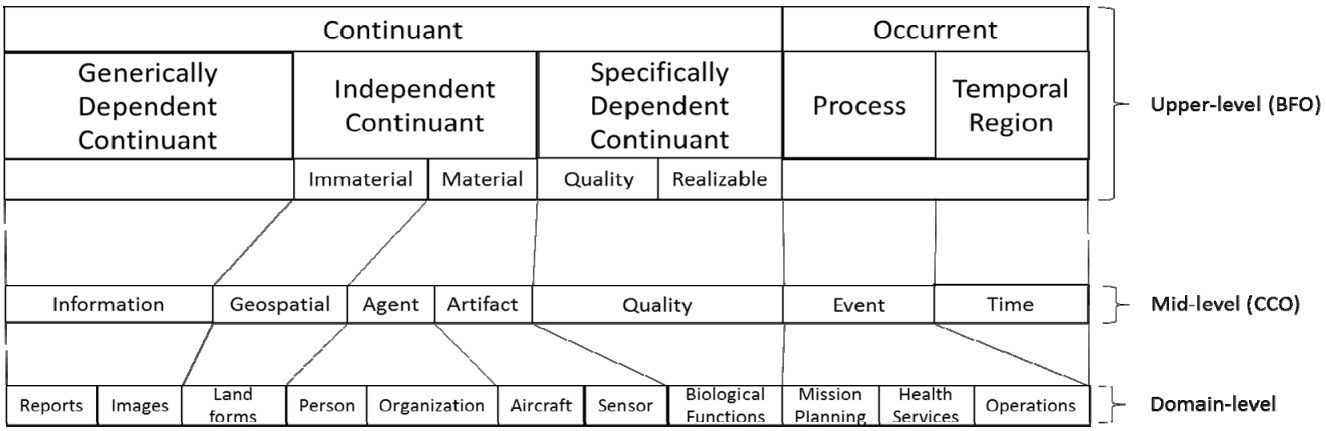

Figure 4 Modular Development of Ontologies

2.5 The benefits of modular ontology development

For present purposes the most important difference between the Modular Ontology and Linked Data methods of ontology development is that in the former ontologies are developed by importing needed content available in pre-existing ontologies rather than re-creating this content anew. BFO and CCO act as seed ontologies for specific domain-level ontologies which in their turn act as seed ontologies for ontologies being developed for other domains. This re-use of existing content has the benefit of reducing effort as there is an ever decreasing need to create new content in the ontologies.

As classes are re-used rather than duplicated there is no need for the kinds of class mappings that are used in Linked Data. There is also a single class hierarchy from any leaf class up to its root class in BFO. Not only does this improve/insure semantic consistency of the schema provided by the ontologies it allows an algorithm to ascend the class hierarchy until it finds a pair of ancestor classes between which a relationship has been asserted and then apply that relation to the original two classes. For example, to the classes of Acoustic Sensor and Acoustic Modality Function, the algorithm would apply the *inheres in* relationship as it is asserted to exist between the classes of Specifically Dependent Continuant and Independent Continuant, which are ancestor classes of Acoustic Modality Function and Acoustic Sensor respectively.

3. BINDING THE ONTOLOGIES TO A NOSQL DATABASE SCHEMA

According to its website[13] Apache HBase™ is the Hadoop database, a distributed, scalable, big data store. As such it is a choice for the data store of a data lake. In this section we describe the HBase™ data model and how the semantics provided by modular ontologies can be bound to this model to function as the at write schema.

[13] http://hbase.apache.org/

3.1 The HBase™ Data Model

HBase™ has a data model described as being a column family model, a variant of the key-value model. In key-value models a key is associated with a record that is composed of a set of values. In a column family model, a key is associated with a set of column families which are composed of sets of column qualifiers. An illustration of the key-value model and the column family model is provided below in Figure 5.

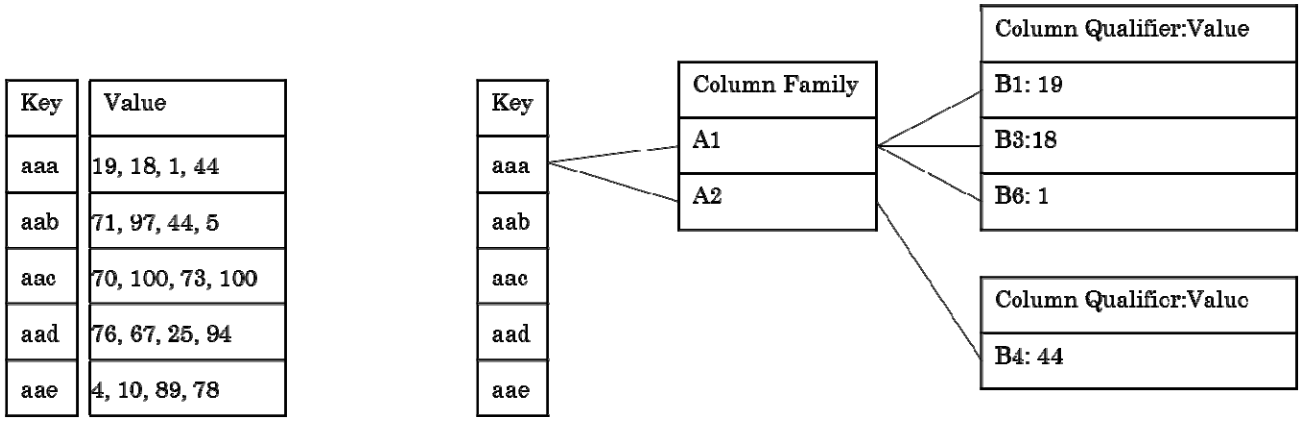

Figure 5 The Key-Value Model vs. the Column Family Model

To illustrate the column family model, in a table storing information about persons, the row key would be a unique identifier of a person, column families would be groups of attributes such as: identifiers, orders and addresses. Within the identifier column family there might be column qualifiers such as name, social security number and customer number. Within the order column family there might n-many column qualifiers, one for each order. Within the address column family there might be column qualifiers of billing address and shipping address. In HBase™ column qualifiers can be timestamped. Continuing the example, the column qualifiers of billing address and shipping address might have multiple instances for different timestamps to accommodate changes of address.

Row keys are sorted lexicographically with the lowest order appearing first and so the choice of key is important to grouping like entities together to optimize scans while distributing rows over the nodes of the cluster to avoid too many reads, writes or other operations being directed at a small number of nodes and causing their performance to become degraded. Column families are containers that group columns into sets to give them separate storage and semantics. The current version of HBase™ under performs with anything above two or three column families. In fact, the reference guide suggests using no more than one column family per table and using another column family only if queries are usually made against one or the other but not both.

3.2 Binding ontology classes to the HBase™ model

We envision a data lake persisted in HBase™ as being divided into tables with each being designed to hold information about a distinct type of entity, examples being persons, organizations, equipment, events and geopolitical entities. Rows in these tables would contain data about instances of these types grouped by column families into different types of attributes whose values would be held in columns differentiated by column qualifiers. While the data, the manner in which queries will be issued against that data and the performance issues and limitations of HBase™ described above will affect the actual binding implementation, the idea is straightforward: simply use class names from the ontologies as the row keys, column family names and column qualifiers.

If we had free rein in the creation of a table schema in an HBase™ database for sensor platforms we would use a unique identifier like a serial number concatenated with the class names for sensor platform and the specific type of platform. The row key of an acoustic sensor would then be SensorPlatform/AcousticSensor/23456789, written in this order to group sensor platforms together. Whether this leads to hotspotting will depend upon the number of sensor platforms but

if it did there are various methods, such as salting[14] that can be used to correct it. Column families seem a natural fit for modeling different views of an entity and it is unfortunate that performance issues limit the number of these. Again, if we had full license we would add column families to the sensor platform table for identifiers, qualities, parts, functions and typical uses, selecting class names from the ontologies as the column family names. Within each of these families there would be column qualifiers named using classes of types of identifiers (e.g. serial number), qualities (e.g. weight), parts (e.g. transducer), functions (e.g. acoustic modality function) and typical uses (e.g. object identification). A notional view of this table schema is given below in Figure 6.

```
{
SensorPlatform/AcousticSensor/23456789: {
        DesignativeInformationContentEntity: {
                ArtifactModelName:"Acme XYZ",
                SerialNumber:"23456789"
                }
        Quality:{
                Weight:"5 kg"
                }
        ArtifactFunction:{
                LineOfBearingDetectionFunction:"True",
                AcousticModalityFunction:"True"
                }
        Detection Event Sensor Process:{
                ActDetectionEventSensorProcess:"True"
                }
        }
}
```

Figure 6 HBase™ schema using class names

The modifications to this schema that are made necessary by the performance issues and limitations of HBase™ are that the row keys, column family names and column qualifiers all need to be shortened and the number of column families need to be reduced. A reversible hashing function or an annotation property in the ontology that acted as an HBase™ label are both options for reducing the length of the string while preserving the link to the ontologies. Even reducing the number of column families to one would not affect the semantics or the translation technique based upon it.

4. TRANSLATING THE SCHEMA TO A GRAPH REPRESENTATION

The inputs to the translation method from the data lake schema a graph representation are row keys and column qualifiers that are named in a manner that is linked to a class name in a set of modular ontologies. Within the modular ontologies, object properties and data properties have specified domains and ranges that are either class names or datatypes. From these inputs the generation of a graph representation requires only the addition of the object property or data property that correctly links the key value class to a column qualifier class. This is accomplished through a single query:

```
SELECT ?property
WHERE {
    ?property rdfs:domain ?row_key_class .
    ?property rdfs:range ?column_qualifier_class .
}
```

If the query returns a single result then the addition is complete.

[14] https://hbase.apache.org/book.html#rowkey.design

If the query returns multiple results this may be due to the properties being subproperties of a parent property. This case is handled by selecting the distinct property for which the multiple properties are subproperties. Another reason for there being multiple results to the query is that the properties are inverses of each other which is particularly common when the row key class and column qualifier class are the same. Examples of this are the object properties that link geospatial entities to one another (e.g. spatial proper part of, has spatial proper part). This case is handled by asserting both relationships in the graph representation.

If the query returns no results then the search is repeated by first ascending the class hierarchy of the column qualifier class as these classes tend to be more specific than the row key classes. If no results are returned the search is repeated by ascending the hierarchy of the row key class and then by ascending both class hierarchies in tandem. Creating an object property with owl:Thing as both domain and range is a simple failsafe for cases in which no other relationships are found. A depiction of how this method translates the schema depicted in Figure 6 into a graph format is provided in Figure 7.

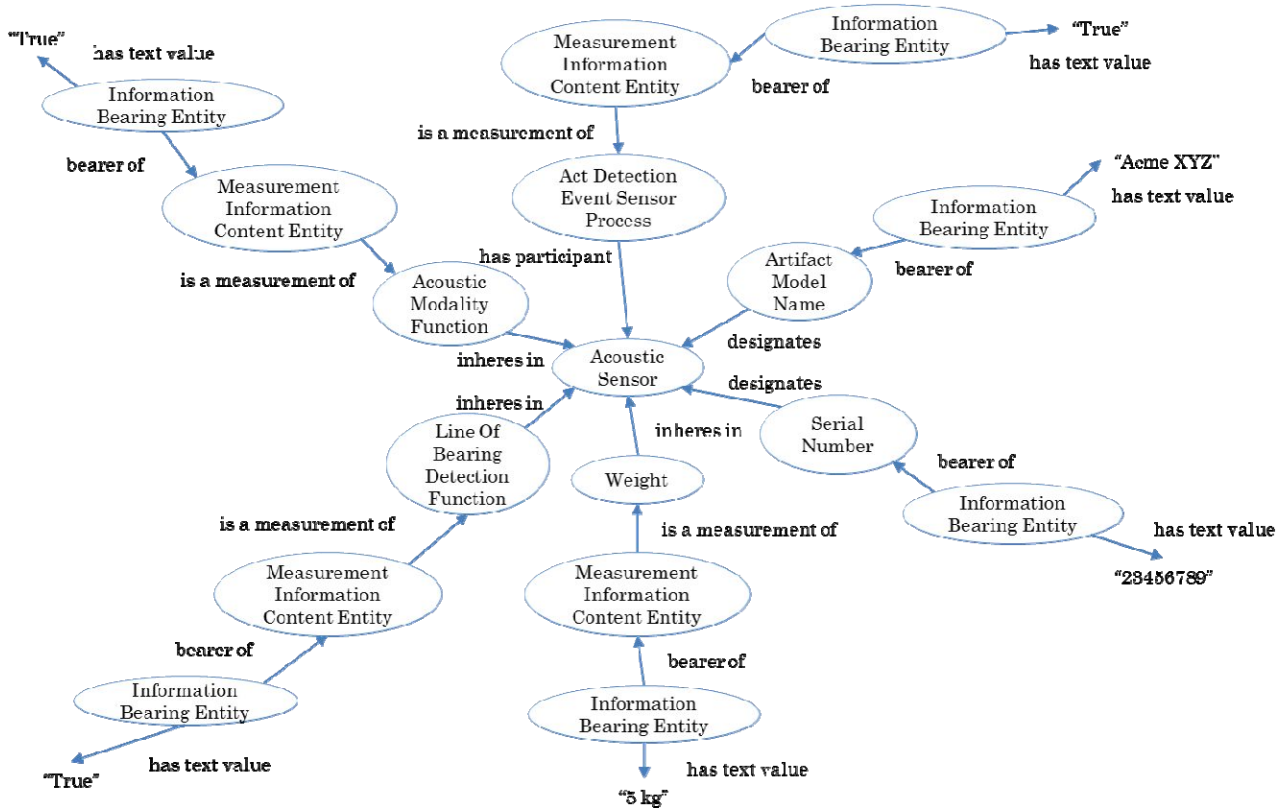

Figure 7 Graph representation of HBase schema

5. FUTURE WORK

We have described a functional design of a method for adding a coherent semantics to the schema of a data lake database and then leveraged that semantics to translate data from a column family model to a graph model. Near term future work will attempt to address two shortcomings of the functional design.

The first of these shortcomings is that some of the content of the graph is redundant. Notice that in the data, a value of "True" is stored to indicate that the sensor platform has an acoustic modality function, a line of bearing detection function and participates in acts of detection. While it is possible that some uses of data would benefit from preserving this value in the graph, possibly for provenance, most would see these text values offering no more information than the

assertion that the instance of the function inheres in the platform or that the instance of the act has the platform as a participant. Methods are needed to edit the translated graph to remove such redundant information if desired.

The second shortcoming is that some of the content of the graph is false. The instance of the artifact model name is asserted to designate the sensor platform. This is the result of the method finding the designates object property as holding between a parent class of the column qualifier (i.e. Designative Information Content Entity) and a parent class of the key value class (i.e. Entity). While not depicted in the example, if data had indicated that the instance of the sensor platform did not have a function by a value of "False", the method is not current capable of preventing a generation of an instance of the function as inhering in the platform.

We are experimenting with methods to auto-correct and edit the graph that include stored SPARQL queries and SHACL rules. Both methods have initially proven to work with parts of the graph generated from column qualifiers that are defined at design time and match existing classes in the ontology. Whether they continue to perform at scale is unknown. The case in which column qualifiers are generated at runtime will require additional research. The initial research has been to use machine learning algorithms to create and correct place a new term into the ontology. If successful, the link between the column qualifier and ontology term would be established and would then fall under the methods described here.

REFERENCES

[1] Terrizzano, I.G., Schwarz, P. M., Roth, M., Colino, J.E., "Data Wrangling: The Challenging Journey from the Wild to the Lake," InCIDR, (2015).

[2] Arp. R., Smith, B., Spear, A.D., "Building Ontologies with Basic Formal Ontology," MIT Press, (2015).