Creation and Interaction with Large-scale Domain-Specific Knowledge Bases

S. Bharadwaj[+] L. Chiticariu[*] M. Danilevsky[*] S. Dhingra[+] S. Divekar[+]
A. Carreno-Fuentes[+] H. Gupta[θ] N. Gupta[θ] S.-D. Han[+] M. Hernández[*]
H. Ho[+] P. Jain[θ] S. Joshi[θ] H. Karanam[θ] S. Krishnan[θ]
R. Krishnamurthy[*] Y. Li[*] S. Manivannan[+] A. Mittal[θ] F. Özcan[*]
A. Quamar[*] P. Raman[+] D. Saha[θ] K. Sankaranarayanan[θ] J. Sen[θ]
P. Sen[*] S. Vaithyanathan[+] M. Vasa[+] H. Wang[+] H. Zhu[*]

[*] **IBM Research-Almaden** [+] **IBM Watson** [θ] **IBM Research-India**

{shreyas.bharadwaj,chiti,mdanile,sdhingra,scdiveka,acarreno,sang-don.han,mahernan,ctho,rajase,yunyaoli,smaniva, fozcan,ahquamar,pchozhi,senp,vaithyan,mitesh.vasa,haowang,huaiyu}@us.ibm.com

{higupta8,ngupta4j,pajain06,salijosh,hkaranam,sarkris5,arakeshk,diptsaha,kartsank,jaydesen}@in.ibm.com

ABSTRACT

The ability to create and interact with large-scale domain-specific knowledge bases from unstructured/semi-structured data is the foundation for many industry-focused cognitive systems. We will demonstrate the Content Services system that provides cloud services for creating and querying high-quality domain-specific knowledge bases by analyzing and integrating multiple (un/semi)structured content sources. We will showcase an instantiation of the system for a financial domain. We will also demonstrate both cross-lingual natural language queries and programmatic API calls for interacting with this knowledge base.

1. INTRODUCTION

Knowledge bases (KBs) populated with facts extracted and integrated from unstructured and/or semi-structured data is the foundation of most modern artificial intelligent systems. Not surprisingly, research on KB construction and interaction has received wide attention from both academia and industry [5, 7, 4] in recent years. Despite the success of popular open-domain KBs such as FreeBase [3] and Yago [9], constructing and querying domain-specific KBs remains an open problem. The key challenges includes: (1) how to represent and incorporate domain knowledge; (2) how to enable a robust platform-agnostic workflow for knowledge extraction and integration from different data sources in different formats in both batch and incremental fashion; (3) how to monitor and maintain data quality; (4) how to store and query the KB to support a wide-spectrum of use cases.

In this demo, we will showcase Content Services system designed to meet these challenges. We will use the constructing and interacting with a KB in the financial domain as a representative use case to illustrate the inherent challenges related to domain-specific KBs.

Proceedings of the VLDB Endowment, Vol. 10, No. 12
Copyright 2017 VLDB Endowment 2150-8097/17/08.

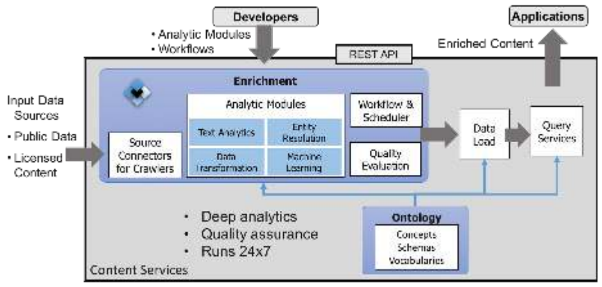


Figure 1: Content Services: Architecture

2. SYSTEM OVERVIEW

Figure 1 depicts the architecture of our *Content Services* system. Content Services are comprised of a number of services divided into three areas: Enrichment, Ontology, and Knowledge Base. These services are deployed on top of a scalable architecture, like Hadoop and Spark, for processing large amounts of structured and unstructured content sources and creating KBs of integrated entities. Each service is exposed to users via REST APIs.

Enrichment Services are in charge of ingesting data from multiple sources and processing that data in workflows of analytic components. These workflows, deployed on top of a scalable architecture such Hadoop and Spark, produce entities (data objects) that are loaded into a KB using the Knowledge Base Service. The Ontology Service is a repository of schemas and vocabularies that describe the entities produced by enrichment, describe intermediate data produced and used by enrichment workflows, and help generate queries over the KB. The Knowledge Base Service, which may utilize Hadoop, RDBMS, or other appropriate data stores stores and indexes the generated entities in the KB and exposes appropriate interfaces to enable user interaction. For the financial application, this includes a natural language interface where users can interact with the KB using natural language queries and an API which enables programmatic access by user applications. The queries expressed using these interfaces are converted into an intermediate query expressed over the domain ontology using the Ontology Query Language (OQL) that we have developed [8]. The query language supports a wide range of queries, including OLAP style, against the KB.

2.1 Enrichment Services

Enrichment Services allows users to register analytic components, which can be combined into a workflow that ingests and uses a domain ontology to transform raw data into an entity-centric KB. Enrichment exposes four types of analytic services: *Annotation* for information extraction, *Data Transformation* for mapping and filtering data, *Entity Resolution* for matching mentions of the same object, and *Machine Learning* for scoring documents based on a previously trained model. The Enrichment Service is designed with the following desiderata in mind.

Enable independent development of components and composition of workflow. Workflows are described by customized declarative specifications that stitch multiple analytics components into a DAG of operations (dataflow description), declare where the input data is located (source description), and describe the conditions needed to start the execution of each component (schedule description).

In the data flow description, each analytic component has a descriptor that specifies its intrinsic properties (such as the runtime engine as well as its required parameter values) and its extrinsic properties (such as the inputs and output connections and their types). The system verifies that the data types between connecting components are compatible. At execution time, when the system needs to run a component, it will consult its descriptor and start the required runtime engines and pass the input data into that engine. The system can be executed in different parallel platforms (e.g. Hadoop/Spark) to achieve parallelism.

Handle continuous incoming data. The schedule description specifies when the run of a component can be triggered, and the type of data window. For example, a component can be executed at certain time every day or week, or when new input data is available. A component may just require the input data to run, or require all past input. As the input data from different sources arrive at different rates and volumes, the Scheduler is responsible for starting various components to run when their conditions are met, and for passing appropriate output to downstream components. In particular, it ensures that each component has complete input data when it runs.

Maintain well defined states. Each flow can be stopped and resumed by user requests (e.g. to accommodate a scheduled machine maintenance). It may also transition into a failure state, and be resumed after the failure is corrected, be resumed, preserving previously completed data.

Monitor data quality. A challenge when dealing with long-running flows is monitoring the quality of the generated data. The Quality Evaluation service allows users to register pre-defined queries that must be run whenever a component is executed. These queries can be specified to profile the generated data through statistical summaries (number of records, field value histograms, etc.) or constraints (values that should not be null, relationships between different fields, etc.). The results of these queries are stored together with the data results and can be queried by users to monitor for changes in expected values and/or trends, and detect data outliers and unexpected behavior.

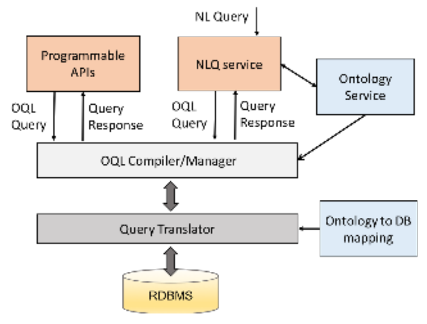


Figure 2: Query Services Architecture

2.2 Query Services

We provide query services over the enriched data via two separate interfaces: a programmable API and Natural Language Query (NLQ). Both interfaces use OQL expressed against the domain-specific ontology. In this section, we briefly introduce OQL and various components that are used in query services (see Figure 2).

Ontology Query Language: In the Content Services system, KBs are associated with ontologies and can be queries with Ontology Query Language (OQL). OQL operates on a set of ontology concepts and relationships, and provides an abstraction for expressing queries independent of the underlying back-end data store.

OQL can express sophisticated queries that include aggregations, unions and nested sub queries among others. Each OQL query block consists of a SELECT clause and a FROM clause, with optional WHERE/GROUOP BY/ORDER BY/FETCH FIRST/HAVING clauses. An OQL query operates on the set of tuples constructed by the Cartesian product of the concepts referred to in the FROM clause.

OQL allows three types of predicates in the WHERE clause: (1) predicates for comparing concept properties with constant values, (2) predicates for expressing joins between concepts and (3) predicates for comparing concept properties with a fixed set of values, potentially computed via a nested query. OQL is also composable and allows arbitrary level of nesting in the SELECT, FROM, WHERE and HAVING clauses. For more details we refer the reader to [8].

Ontology-to-Database Mapping: The enriched data is transformed into a relational format and stored in a RDBMS. For every concept in the ontology and its data properties, one or more relational tables are created. For functional relationships in the ontology, primary key and foreign key constraints are generated. Attributes which need to be indexed in the database are annotated in the ontology. The enriched data, which is in JSON format, is first transformed to CSV using Spark SQL, and then loaded into the RDBMS. We also keep track of the *ontology-to-database mapping*, which

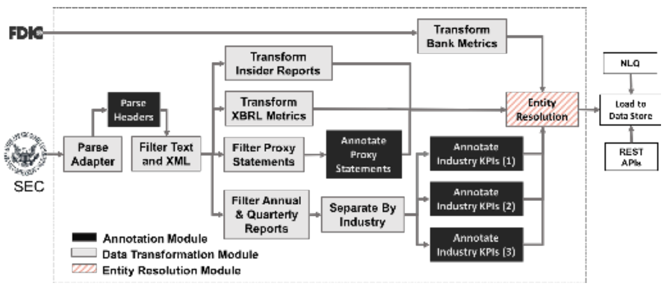


Figure 3: Financial Flow.

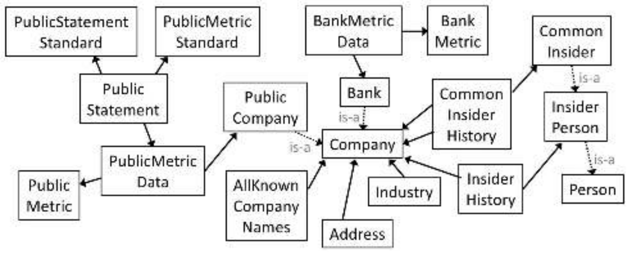


Figure 4: Financial Ontology.

describes how the ontology elements (concepts, properties, and relations) are mapped to database objects (e.g., tables, views, columns, and referential integrity constraints).

Translation Index: Translation Index (TI) is essentially a semantic index of key properties of concepts that require disambiguation. It generates variants for each data property value in the KB, and maintains a map that allows reverse look-up for the values. In the context of financial domain, TI indexes properties such as *company.name*, *person.name*, *metric.name*, *insider.title* etc. and generates variants for each such property value. For example, given a person name is 'John K. Doe', TI generates variants such as 'John Doe', 'John D.', 'J. Doe', 'John', 'Doe' and so on. Variants are generated using a combination of techniques such as mapping rules, dictionaries, and regexes. When a user query contains any of these variants, TI responds with a match and confidence. Confidence depends on the nature of variant generation and ranges between 'high', 'medium' and 'low', so 'John Doe' gets a high confidence whereas 'John' gets a low confidence. The response contains not just the original value 'John K. Doe' but also the ontology URI for the corresponding property.

Query Translator: The *Query Translator* takes an OQL query constructed against the domain ontology as input, along with the ontology itself and the corresponding Ontology-to-Database Mapping. The Query Translator then generates the corresponding SQL queries to run against the RDBMS and returns the response in JSON format.

There are more than one mapping from a given ontology to a relational schema, and the query translator handles different mappings using the ontology-to-database mappings. For example, a child concept in an inheritance hierarchy could result in a child table in the relational schema, and the attributes that are inherited from the parent can either be stored in the parent table or replicated in the child table. In the former case, the translator inserts the necessary join with the parent table to pick up the attributes, if needed. You can find more details about the query translator in [8].

3. FINANCIAL CONTENT KB

Figure 3 depicts an instantiation of the Enrichment Flow of Content Services for the financial domain. The flow ingests public filings from SEC and FDIC (in XBRL and HTML), transforms the data via a series of analytics modules, and constructs a KB with facts about financial entities and their relationships, similar in spirit to [2]. Note that this is not a trivial linear flow of simple operations.

Figure 4 shows a pre-defined ontology, designed by domain experts, that drives the population of financial entities and their relationships. For example, both banks and public companies have related metric information, but only Public Metrics are additionally identified as belonging to particular statements (e.g., Income Statement), thus necessitating additional concepts such as `PublicStatement`. As another example, there should only be one instance for a yearly metric for a company in any given year.

The input data for this flows is in different formats and arrives from two sources at different frequencies and volumes. Each component of the workflow is a complex analytic operation (more than simple filters and joins operations in ETL-like workflows) that deals, in many case, with large data sets and, thus, must be executed on different scalable platforms (e.g. Hadoop/Spark). Since data is arriving at regular intervals, the workflow must be ready to start processing new data as it arrives. Moreover, each component of the workflow might have a different trigger. Most component in the workflow can start executing when a new *batch* of data appears on its input side, process that batch alone, and produce an output batch. However, the last component in our flow does entity resolution, an aggregate operation that require all input batches, not just the most recent one. If this workflow fails at certain point, it must be able to resume and recover from previous failure.

The resulting Financial Content KB consists of $20,000$ US public companies and $10,000$ banks, $180,000$ insiders, and 96-million and 480-million unique metric data points for public companies and banks respectively.

4. INTERACTION WITH THE KB

4.1 Cross-lingual Natural Language Query

The NLQ component translates the input natural language query into OQL, which is executed by Query Services.

The NLQ component includes three parts: (i) Ontology Evidence Annotator (OEA) that takes the input query and map the tokens to the elements of the domain ontology or to the instances in the KB along with the corresponding ontology elements ("Alibaba" $\rightarrow$ Company.name) using Translation Index and semantic constraints ("Show me investments in Alibaba" $\rightarrow$ Prefer "Alibaba" as Investee as compared to Investor); (ii) Ontology-driven Interpretations Generator that maps all the evidences produced by OEA, and to Ontology graph to produce ranked list of interpretations and (iii) Ontology Query Builder that translates each such interpretation into the intermediate OQL query.

NLQ can take input queries in different languages to query the underlying monolingual KB. OEA addresses all the linguistic challenges related to the query by generating a language-agnostic evidence set. To do so, OEA relies on (a) Multilingual Semantic Role Labeling [1] that helps parsing the query tokens and identifying semantic constraints and (b) Multilingual Translation Service that maps query tokens in the input language to an ontology element [1].

4.2 Programmable API

The programmatic API enables developers to interact with the KB without explicit knowledge of the KB schema, while building useful applications from the content. The parameterized API calls generate OQL queries, which are then executed against the runtime, and the response is returned in JSON format.

4.3 Rich Entities

The entity-centric views provided by ontology driven KBs enable querying for individual real-world entities, as well as exploring entity information (such as the address or most recent net revenue of a company) through explicit querying. In practice, however, users must often resort to issuing many point queries in order to gather all relevant and context-specific information about a given entity.

To enhance user experience and ameliorate the problem of the user needing to issue many queries, we propose the concept of *Rich Entities*. These rich entities comprise all the relevant and context-specific information for given real-world entities, and serve as efficient and meaningful responses to user queries against these entities in a KB. The metadata that describes the structure and composition of a rich entity, such as Company or Insider, can be viewed in terms of a predefined subgraph of the domain ontology graph, and therefore a predefined OQL query that is executed at runtime to retrieve the Rich Entity information. Figure 6 shows an example Insider Rich Entity , presenting the name, current job title, and recent employment history in one visualization.

5. SYSTEM DEMONSTRATION

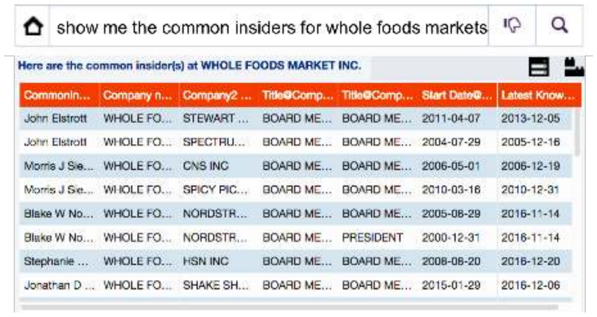
The demo will showcase different query classes supported by the Financial Content KB via a web based natural language interface as well as a programmatic REST based API with the following demo scenarios.

Demo Scenario 1: The first step to build an effective board of directors is to find the right candidates. Imagine that we are looking for people with the following profile to seat on the board for a start-up providing healthy kids' lunches: Retail grocery store industry experience, familiar with children-related product/services, proven track record as a board member, and bandwidth to serve on the board.

The search for such a candidate would require us to issue a set of queries against the KB as described below.

Q1: Show me the common insiders for Whole Foods Markets The results (Figure 5) list people on the board of Whole Foods Markets, a popular US-based retail grocery store company, and also gives us information about the other companies that they have been associated with while on the board of Whole Food Markets. As can be see, `Linda A. Mason` has also served on the Board of Bright Horizons Family Solutions, a nation-wide child-care provider. Thus she also meets the second requirement of being familiar with children-related product/services.

[1]This dictionary can be manually provided or populated semi-automatically [6].

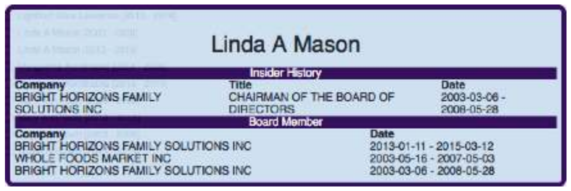


show me the common insiders for whole foods markets

Here are the common insider(s) at WHOLE FOODS MARKET INC.

Commonin...	Company n...	Company2 ...	Title@Comp...	Title@Comp...	Start Date@...	Latest Know...
John Elstrott	WHOLE FO...	STEWART ...	BOARD ME...	BOARD ME...	2011-04-07	2013-12-05
John Elstrott	WHOLE FO...	SPECTRU...	BOARD ME...	BOARD ME...	2004-07-29	2005-12-16
Morris J Sie...	WHOLE FO...	CNS INC	BOARD ME...	BOARD ME...	2006-05-01	2006-12-19
Morris J Sie...	WHOLE FO...	SPICY PIC...	BOARD ME...	BOARD ME...	2010-03-16	2010-12-31
Blake W No...	WHOLE FO...	NORDSTR...	BOARD ME...	BOARD ME...	2005-08-29	2016-11-14
Blake W No...	WHOLE FO...	NORDSTR...	BOARD ME...	PRESIDENT	2000-12-31	2016-11-14
Stephanie ...	WHOLE FO...	HSN INC	BOARD ME...	BOARD ME...	2008-08-20	2016-12-20
Jonathan D ...	WHOLE FO...	SHAKE SH...	BOARD ME...	BOARD ME...	2015-01-29	2016-12-06

Figure 5: Common Insiders with Whole Foods.



Linda A Mason

Insider History

Company	Title	Date
BRIGHT HORIZONS FAMILY SOLUTIONS INC	CHAIRMAN OF THE BOARD OF DIRECTORS	2003-03-06 - 2008-05-28

Board Member

Company	Date
BRIGHT HORIZONS FAMILY SOLUTIONS INC	2013-01-11 - 2015-03-12
WHOLE FOODS MARKET INC	2003-05-16 - 2007-05-03
BRIGHT HORIZONS FAMILY SOLUTIONS INC	2003-03-06 - 2008-05-28

Figure 6: Rich Entity for Linda Mason.

Q2: Which company boards has Linda Mason served on in the last 10 years Figure 6 shows the rich entity associated with Linda Mason providing her employment history including the key positions that she has held across all companies in the recent past.

Demo Scenario 2: We will also showcase the Financial Content API that enables programmatic access to the KB by user applications. In the demo, we will show a Swagger interface (omitted in the interest of space) where the users can interact with the API and provide appropriate parameters for generating different query templates. The users will be able to see the query template, the OQL query generated and the JSON response to the query received from the KB.

6. REFERENCES

[1] A. Akbik and Y. Li. Polyglot: Multilingual semantic role labeling with unified labels. *ACL (Demo)*, pages 1–6, 2016.

[2] S. Balakrishnan and et al. Midas: Integrating public financial data. In *Proc. SIGMOD*, pages 1187–1190, 2010.

[3] K. Bollacker, R. Cook, and P. Tufts. Freebase: A shared database of structured general human knowledge. In *Proc AAAI*, volume 2, pages 1962–1963, 2007.

[4] D. A. Ferrucci and et al. Building Watson: An overview of the DeepQA project. *AI Magazine*, 31(3):59–79, 2010.

[5] T. Mitchell and et al. Never-ending learning. In *AAAI*, 2015.

[6] F. J. Och and H. Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003.

[7] C. D. Sa and et al. Deepdive: Declarative knowledge base construction. *SIGMOD Record*, 45(1):60–67, 2016.

[8] D. Saha and et al. Athena: An ontology-driven system for natural language querying over relational data stores. *Proc. VLDB Endow.*, 9(12):1209–1220, Aug. 2016.

[9] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.