VKGBuilder – A Tool of Building and Exploring Vertical Knowledge Graphs

Tong Ruan, Haofen Wang, and Fanghuai Hu

East China University of Science & Technology, Shanghai, 200237, China
{ruantong,whfcarter}@ecust.edu.cn, xiaohuqi@126.com

Abstract. Recently, search engine companies like Google and Baidu are building their own knowledge graphs to empower the next generation of Web search. Due to the success of knowledge graphs in search, customers from vertical sectors are eager to embrace KG related technologies to develop domain specific semantic platforms or applications. However, they lack skills or tools to achieve the goal. In this paper, we present an integrated tool VKGBuilder to help users manage the life cycle of knowledge graphs. We will describe three modules of VKGBuilder in detail which construct, store, search and explore knowledge graphs in vertical domains. In addition, we will demonstrate the capability and usability of VKGBuilder via a real-world use case in the library industry.

1 Introduction

Recently, an increasing amount of semantic data sources are published on the Web. These sources are further interlinked to form Linking Open Data (LOD). Search engine companies like Google and Baidu leverage LOD to build their own semantic knowledge bases (called *knowledge graphs*[1]) to empower semantic search. The success of KGs in search attracts much attention from users in vertical sectors. They are eager to embrace related technologies to build semantic platforms in their domains. However, they either lack skills to implement such platforms from scratch or fail to find sufficient tools to accomplish the goal.

Compared with general-purpose KGs, knowledge graphs in vertical industries (denoted as VKG) have the following characteristics: a) *More accurate and richer data* of certain domains to be used for business analysis and decision making; b) *Top-down construction* to ensure the data quality and stricter schema while general KGs are built in a bottom-up manner with more emphasis on the wide coverage of data from different domains; c) *Internal data stored in RDBs* are further considered to be integrated into VKGs; and d) Besides search, VKGs should provide *user interfaces* especially for KG construction and maintenance.

While there exist tool suites (e.g., LOD2 Stack[2]) which help to build and explore LOD, these tools are mainly developed for researchers and developers of the Semantic Web community. Vertical markets, on the other hand, need

[1] http://en.wikipedia.org/wiki/Knowledge_Graph

[2] http://lod2.eu/

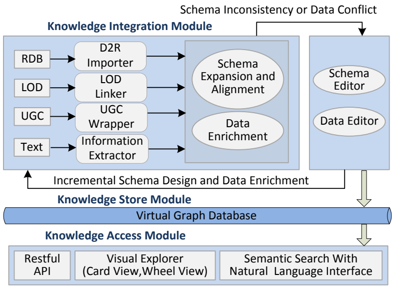
Schema Inconsistency or Data Conflict
Knowledge Integration Module
RDB
LOD
UGC
Text
D2R Importer
LOD Linker
UGC Wrapper
Information Extractor
Schema Expansion and Alignment
Data Enrichment
Schema Editor
Data Editor
Incremental Schema Design and Data Enrichment
Knowledge Store Module
Virtual Graph Database
Knowledge Access Module
Restful API
Visual Explorer (Card View,Wheel View)
Semantic Search With Natural Language Interface

Fig. 1. Architecture of VKGBuilder

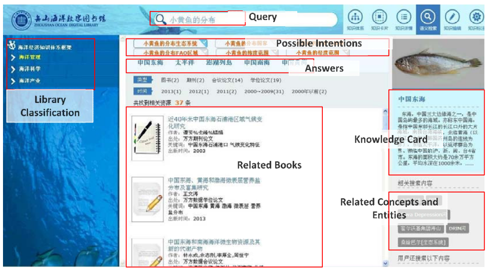

Fig. 2. Semantic Search Interface

end-to-end solutions to manage the life cycle of knowledge graphs and hide the technical details as much as possible. To the best of our knowledge, we present the first suitable tool for vertical industry users called VKGBuilder. It allows *rapid and continuous VKG construction* which imports and extracts data from diverse data sources, provides a mechanism to detect intra- and inter-data source conflicts, and consolidates these data into a consistent KG. It also provides *intuitive and user-friendly interfaces* for novice users with little knowledge of semantic technologies to understand and exploit the underlying VKG.

2 Description of VKGBuilder

VKGBuilder is composed of three modules namely the *Knowledge Integration* module, the *Knowledge Store* module, and the *Knowledge Access* module. The whole architecture is shown in Figure 1. Knowledge Integration is the core module for VKG construction with three main components. Knowledge Store is a virtual graph database which combines RDBs, in-memory stores and inverted indexes to support fast access of VKG in different scenarios, and the Knowledge Access module provides different interfaces for end users and applications.

2.1 Knowledge Integration Module

– *Data Importers and Information Extractors.* Structured data from internal relational database are imported and converted into RDF triples by D2R importers[3]. A *LOD Linker* is developed to enrich VKG with domain ontologies from the public linked open data. For the user generated contents (UGCs), we mainly consider encyclopaedic sites like Wikipedia, Baidu Baike, and Hudong Baike. Due to the semi-structured nature of these sites, *wrappers* automatically extract properties and values of certain entities. As for unstructured text, distant-supervised learning methods are adapted to discover missing relations between entities or fill property values of a given entity where the above extracted semantic data serve as seeds.

[3] `http://d2rq.org/`

- *Schema Inconsistency and Data Conflict Detection.* After semantic data are extracted or imported from various sources, data integration is performed to build an integrated knowledge graph. During integration, schema-level inconsistency and data-level conflicts might occur. Schema editing is used to define axioms of properties such as (e.g., functional, inverse, transitive), concept subsumptions, and concepts of entities. Then a rule-based validator is triggered to check whether the newly added data or imported ontologies will cause any conflicts with existing ones. The possible conflicts are resolved by user defined rules or delivered to domain experts for human intervention.
- *Schema and Data Editor.* Knowledge workers can extend or refine a VKG in both schema-level and data-level with a collaborative editing interface.

2.2 Knowledge Access Module

- *Visual Explorer.* It includes three views namely the *Wheel View*, the *Card View*, and the *Detail View*. The Wheel View organizes concepts and entities in two wheels. In the left wheel, the node of interest is displayed in the center. If it is a concept, its child concepts are neighbors in the same wheel. If it is an entity, its related entities are connected via properties as outgoing (or incoming) edges. When a related concept (or entity) is clicked, the right wheel is expanded with the clicked node in the center surrounded with its related information on the VKG. Thus, we allow users to navigate through the concept hierarchy and traverse between different entities. The Card View visualizes entities in a force-directed graph layout, which is similar to the galaxy visualization in a 3D space. The Card View also allows to change the focus through drag and drop as well as zoom-in and zoom-out. The Detailed View shows all properties and property values of a particular entity. The three views can be switched from one to another in a flexible way.
- *Semantic Search with Natural Language Interface.* Users can submit any keyword query or natural language question. The query is interpreted into possible SPARQL queries with natural language descriptions. Once a SPARQL query is selected, the corresponding answers are returned, along with relevant documents which contain semantic annotations on these answers. Besides, a summary (a.k.a, knowledge card) of the main entity mentioned in the query or the top-ranked answer is shown. Related entities defined in the VKG as well as correlated entities in the query log are recommended.
- *Restful APIs.* They are designed for developers with little knowledge of semantic technologies to access the VKG using any programming language from any platform at ease. These APIs are actually manipulations of SPARQL queries to support graph traversal or sub-graph matching on the VKG.

3 Demonstration

VKGBuilder is first used in the ZhouShan Library. The current VKG (marine-oriented KG) contains more than 32,000 fishes and each fish has more than 20

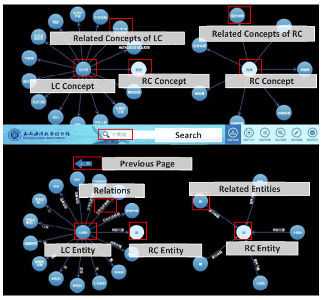

Fig. 3. Wheel View

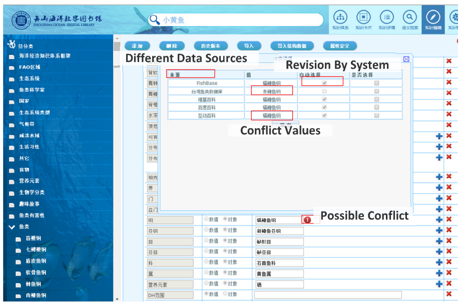

Fig. 4. Conflict Resolution

properties. Besides fishes, VKGBuilder also captures knowledge about fishing grounds, fish processing methods, related researchers and local enterprises. An online demo video of VKGBuilder can be downloaded at `http://202.120.1.49:19155/SSE/video/VKGBuilder.wmv`.

Figure 2 shows a snapshot of the semantic search interface. When a user enters a query “Distribution of Little Yellow Croaker”, VKGBuilder first segments the query into “Little Yellow Croaker” and “Distribution”. Here, “Little Yellow Croaker” is recognized as a fish, and properties about “distribution” are returned. Then all sub-graphs connecting the fish with each property are found as possible SPARQL query interpretations of the input query. Top interpretations whose scores are above a threshold are returned with natural language descriptions for further selection. Once a user selects a query, the answers (e.g., China East Sea) are returned. Also, related books with these answers as semantic annotations are returned. The related library classification of these books are displayed in the left, and the knowledge card as well as related concepts and entities of Little Yellow Croaker are listed in the right panel.

In Figure 3, the Wheel View initially shows the root concept (owl:Thing) in the center of the left wheel (denoted as LC). When a sub-concept `Fish` is clicked, it becomes the center of the right wheel (denoted as RC) with its child concepts (e.g., `Chondrichthyes`). We can also navigate between entities. For instance, `selenium` is one of the nutrients of Little Yellow Croaker. When clicking `selenium`, all fishes containing this nutrient are shown in the right wheel.

The user experience heavily depends on the quality of the underlying VKG. The extraction and importing are executed automatically in the back-end while we provide a user interface for conflict resolution. For “Little Yellow Croaker”, we extract `Ray-finned Fishes` and `Actinopterygii` from different sources as values of the property `Class` in the scientific classification. Since `Class` is defined as a functional property and the two values do not refer to the same thing, a conflict occurs. As shown in Figure 4, VKGBuilder accepts `Actinopterygii` as the final value because this value is extracted from more trusted sources.

Acknowledgements This work is funded by the National Key Technology R&D Program through project No. 2013BAH11F03.