Interlinking Unstructured and Structured Knowledge in an Integrated Framework

Francesco Corcoglioniti, Marco Rospocher, Roldano Cattoni, Bernardo Magnini, Luciano Serafini
FBK-irst
Via Sommarive, 18 - Trento, Italy (I-38123)
Email: {corcoglio,rospocher,cattoni,magnini,serafini}@fbk.eu

Abstract—Despite the widespread diffusion of structured data sources and the public acclaim of the Linked Open Data initiative, a preponderant amount of information remains nowadays available only in unstructured form, both on the Web and within organizations. While different in form, structured and unstructured contents speak about the very same entities of the world, their properties and relations; still, frameworks for their seamless integration are lacking. In this paper we describe the design of the KnowledgeStore, a scalable, fault-tolerant, and Semantic Web grounded storage system for interlinking structured and unstructured data. We discuss its capability to adapt to new types of content and application scenarios, and to provide reasoning and semantic queries services on top of stored contents. We also comment on its envisaged use in the NewsReader EU project to manage large amounts of economical-financial data.

I. INTRODUCTION

The rate of growth of digital data and information is nowadays continuously increasing. While the recent advances in Semantic Web Technologies (e.g., the Linked Data initiative), have favoured the release of large amount of data and information in structured machine-processable form (e.g., RDF dataset repositories), a huge amount of content is still available and distributed through websites, company internal Content Management System (CMS) and repositories, in an unstructured form, for instance as textual document, web pages, and multimedia material (e.g., photos, diagrams, videos). Indeed, as observed in [1], unstructured data accounts for more than 90% of the digital universe.

Although bearing a clear dichotomy for what concern their form, the content of structured and unstructured resources is far from being separated: they both speak about entities of the world (e.g., persons, organizations, locations, events), their properties, and relations among them. Indeed, coinciding, contradictory, and complementary facts about these entities could be available in structured form, unstructured form, or both. Therefore, partially focusing on the content distributed in only one of these two forms may not be appropriate, as complete knowledge is a requirement for many applications, especially in situations where users have to make (potentially critical) decisions. Moreover, some applications inherently require considering both types of content: an example is question answering, where often a user query can only be answered by combining information in structured and unstructured sources.

Despite the last decades achievements in natural language and multimedia processing, now supporting large scale extraction of knowledge about entities of the world from unstructured digital material, frameworks enabling the seamless integration and linking of knowledge coming both from structured and unstructured content are still lacking.

In this paper we describe the design of the KnowledgeStore, a framework that contributes to bridge the unstructured and structured worlds, enabling to jointly store, manage, retrieve, and semantically query, both typologies of contents. Fig. 1 shows schematically how the KnowledgeStore manages these contents in its three representation layers. On the one hand (and similarly to a file system) the resource layer stores unstructured content in the form of resources (e.g., news articles, multimedia files...), each having a textual or binary representation and some descriptive metadata. Information stored in this level is typically noisy and redundant, with the same piece of information potentially represented in different ways in multiple resources. On the other hand, the entity layer is the home of structured content, that based on Knowledge Representation and Semantic Web best practices consists of (subject, predicate, object) statements, which describe the entities of the world (e.g., persons, locations, events), and for which additional metadata is kept to track their provenance and to denote the formal contexts where they hold (e.g., in terms of time, space, point of view). Differently from the resource layer, the entity layer aims at providing a formal and concise representation of the world, abstracting from the many ways it can be encoded in natural language or in multimedia, and thus allowing the use of automated reasoning to derive new statements from asserted ones. Between the aforementioned two layers is the mention layer. It indexes mentions, i.e., snippets of resources (e.g., some characters in a text document, some pixels in an image) that denote something of interest, such as an entity or a statement of the entity layer. Mentions can be automatically extracted by natural language and multimedia processing tools, that can enrich them with additional attributes about how they denote their referent (e.g., with which name, qualifiers, “sentiment”). Far from being simple pointers, mentions present both unstructured and structured facets (respectively snippet and attributes) not available in the resource and entity layers alone, and are thus a valuable source of information on their own.

Thanks to the explicit representation and alignment of
investigated in [3] and tested in the scope of the LiveMemories project. However, we highly revised the design of the previous version, introducing significant enhancements: the new version of the KnowledgeStore, currently under implementation, supports (i) the storing and reasoning on events and related information, such as event relations (the previous version was limited to mentions and entities referring to persons, organizations, geo-political entities, and locations), (ii) scaling on a significantly larger collection of resources (potentially, billions of documents versus a few hundreds of thousands), and (iii) a semantic queries mechanism over its content (no reasoning services was previously offered).

The paper is organized as follows. In Section II, we discuss the role of the KnowledgeStore in applications dealing with both structured and unstructured contents, referring to a concrete motivating scenario (the NewsReader EU project). In Section III we present in details the KnowledgeStore data model, describing its roots in Semantic Web technologies and its capabilities to adapt to different usage scenarios. In Section IV we illustrate the kinds of interfaces through which tools and applications can interact with the KnowledgeStore, while in Section V we describe the KnowledgeStore architecture, presenting each module composing the framework and the physical implementation of the data model. In Section VI we briefly overview related state-of-the-art approaches. Section VII concludes with some final remarks.

II. THE ROLE OF THE KNOWLEDGESTORE IN HETEROGENEOUS-CONTENT APPLICATIONS

We present a concrete scenario, representative of applications processing both structured and unstructured content, that highlights the need for a system such as the KnowledgeStore. The goal of the NewsReader Project is to process daily economical and financial news in order to extract events (i.e., what happened to whom, when and where – e.g., “The Black Tuesday, on 24th of October 1929, when United States stock market lost 11% of its value”), and to organize these events in coherent narrative stories, combining new events with past events and background information. These stories are then offered to professional decision-makers, that by means of visual interfaces and interaction mechanisms will be able to exploit them, exploiting their explanatory power and their systematic structural implications, to make well-informed decisions. Achieving these challenging goals requires:

- to process document resources, detecting mentions of events, event participants (e.g., persons, organizations), locations, time expressions, and so on;
- to link extracted mentions with entities, either previously extracted or available in some structured domain source, and coreferring mentions of the same entity;
- to complete entity descriptions by complementing extracted mention information with available structured knowledge (e.g., DBPedia, corporate databases);

III. THE KNOWLEDGESTORE CONTENT MODEL

The KnowledgeStore content model is based on technologies compliant with the deployment in distributed hardware settings, like clusters and cloud computing. The idea behind the KnowledgeStore was preliminary

---

1 Fulfilling this request involves: (i) to reason in the structured part about which events “Barack Obama” participated to that are of type “sport event”, and identify the corresponding participation statements; (ii) to exploit the links to the mentions those statements have been extracted from; and (iii) to exploit the linking between those mentions and the resources containing them.

2 dbpedia:United_Nations

3 http://www.livememories.org/

4 http://www.newsreader-project.eu/

5 http://dbpedia.org/
to interrelate entities (events and their participants, in particular) to support the construction of narrative stories;
• to reason over events to check consistency, completeness, factuality and relevance;
• to store all this huge quantity of information\(^6\) (on resources, mentions, entities) in a scalable way, enabling efficient retrieval and intelligent queries;
• to effectively offer narrative stories to decision makers.

Note that such requirements, though arisen from the specific application scenario considered within the NewsReader project, are quite typical in many application contexts where enhanced applications (e.g., decision support systems, information retrieval systems, semantic search engines, query answering applications) have to deal with both unstructured content and structured knowledge.

A framework like the KnowledgeStore can effectively contribute to address such kind of requirements. First, the KnowledgeStore allows to store in its three interconnected layers all the typologies of content that have to be processed and produced when dealing with unstructured content and structured knowledge: (i) (annotated) unstructured resources (e.g., financial news) are organized in the resource layer; (ii) the mention layer identifies fragments of resources denoting entities (e.g., a take-over event), relation between entity mentions (e.g., event participation), numerical quantities (e.g., a share price); and (iii) the entity layer stores the structured descriptions of those entities extracted from resources and merged with available structured knowledge (e.g., Linked Data sources, corporate databases).

Second, as shown in Fig. 2, the KnowledgeStore acts as a shared data space supporting the interaction of the several modules and tools envisaged according to the aforementioned requirements: modules retrieve their input data from the KnowledgeStore, and store the results of their processing back in it, so that they can be picked up by other modules. Modules can be roughly classified in 5 categories:

- **Structured and Unstructured content populators**. These modules enable the bulk loading of structured and unstructured contents in the KnowledgeStore.
- **Resource Processors**. These modules work at the resource layer, and take care of performing a pre-processing of the text document, enriching it with linguistic annotations.
- **Mention Processors**. These modules work at the resource and mention layers, exploiting the results of resource processors to instantiate mentions via named entity and event recognition, semantic role labelling, and so on.
- **Entity Processors**. These modules work at the mention and entity layers, exploiting the results of mention processors to instantiate, link, or enrich entities performing tasks such as coreference and knowledge fusion.
- **Applications**. Several modules (e.g., decision support system) can queries the KnowledgeStore—mainly the entity layer (although queries may also require to retrieve documents and mentions)—to obtain information about the entities of the application domain.

### III. The KnowledgeStore Data Model

The KnowledgeStore data model is depicted in the UML class diagram of Fig. 3, and is centred around the three resource, mention and entity layers. Resources and mentions are described using a configurable set of types, attributes and relations. Entities are described with an open set of statements enriched with metadata attributes (e.g., for context and provenance). By design, types, attributes and relations are identified via URIs (hereafter denoted with qualified names and a KnowledgeStore default namespace\(^7\)) and the model is assimilable to an OWL 2 ontology [4], thus allowing to encode both its definition and instance data in RDF [5], and to manipulate them using Semantic Web technologies.

Flexibility is a key requirement for the data model, as (i) different kinds of unstructured and structured content can be stored in different KnowledgeStore instances; and (ii) the kind of information stored in a KnowledgeStore instance may evolve in time (i.e., a fixed schema is not possible). For this reason, the data model is divided in a *fixed part*, embodied in the implementation and kept as small as possible, and a *configurable part* that is specific to each KnowledgeStore instance and is used to organize and fine tune its storage layout.

More in details, the fixed part includes:

- the **Resource**, **Mention** and **Entity** classes and their **dc:identifiers**, which are assigned by the system when an object is created and are then immutable;
- the **Statement** class with essential attributes and subclasses for encoding entity types, attributes and relations;
- the files storing resource representations and their metadata managed by the system (nie:isStoredAs attribute, nfo:FileDataObject class);
- relations **dc:isPartOf** and **refersTo** linking a mention to the containing resource and the entity it possibly denotes;
- relation **extractedFrom** linking a statement to the mention(s) it has been extracted from, used to track provenance and debug information extraction pipelines built on top of the KnowledgeStore.

---

\(^6\)Consider that ~25\% of the daily news deals with finance and economy.

\(^7\)http://dkm.fbk.eu/ontologies/knowledgestore/\#
The formal specification of these elements is provided by a KnowledgeStore ontology that reuses terms from Dublin Core (dc:*), the Nepomuk Information Element vocabulary (nie:*), and the Nepomuk File Ontology (nfo:*)\(^9\). Another OWL 2 ontology, specific to each KnowledgeStore instance, specifies the configurable part of the data model by refining the TBox definitions of the KnowledgeStore ontology. It defines:

- the subclass hierarchy of Resource and Mention;\(^11\)
- additional attributes of Resource, Mention, Statement and their subclasses;
- additional relations among resources or among mentions;
- enumerations and classes used as attribute types (similarly to nfo:FileDataObject);
- restrictions on fixed part relations (not shown in figure).

As an example, Fig. 4 shows how the KnowledgeStore data model has been configured for a complex scenario such as the NewsReader one. News and news annotations from NLP tools are stored at the resource level, each described by a number of metadata from the Dublin Core and Nepomuk vocabularies. Several types of mentions are stored, which denote either an entity (e.g., person, organization, events), a relation among entities (e.g., causal, temporal or subordinate links among event mentions and/or time expressions, derived from the TimeML standard [6]) or a numerical quantity. The NLP Interchange Format (NIF) vocabulary\(^12\) has been used to define basic mention properties, thus enabling interoperability with tools consuming NIF data; in addition, several specific attributes have been added to store information extracted from

\(^{9}\)http://dublincore.org/documents/dcmi-terms/
\(^{10}\)http://www.semanticdesktop.org/ontologies/nie/
\(^{11}\)As the characterization of entities is performed through statements, there is no provision for specific entity subclasses, attributes or relations.

\(^{12}\)http://nlp2rdf.org/nif-1-0
NLP processing. At the entity level, provenance, confidence and contextual metadata have been associated to statements, exploiting also the Simple Event Model (SEM) vocabulary [7].

It is worth noting that the choice of rooting the data model in OWL 2 and using an OWL 2 ontology for its configuration provides a number of benefits. First, it allows sharing stored data on the Semantic Web, e.g., by publishing it as Linked Open Data (a feature that we plan to directly support in the KnowledgeStore). Second, inference and data validation can be performed using an OWL 2 reasoner, although scalability considerations will restrict them on a per-resource or per-mention basis (joint inference on billions of OWL 2 object descriptions is unfeasible). In particular, expressive data validation can be achieved by declaring complex integrity constraints as OWL 2 axioms. In this case, the open world assumption (OWA) underlying OWL 2 and its rejection of the unique name assumption (UNA) must be taken into consideration, as they are designed for inferring missing information rather than detecting constraint violations. UNA can be assumed for the objects managed by the KnowledgeStore, and can be achieved by automatically declaring their identifiers as owl:differentFrom each other. OWA may be inappropriate for certain attributes for which complete information is available, in which case OWL 2 extensions realizing a restricted closed world assumption can be adopted, such as [8], [9].

IV. The KnowledgeStore Interfaces

The KnowledgeStore offers a number of interfaces through which external clients may programatically access and manipulate stored data. In order to define those interfaces in a comprehensive and general way, several users (most of them involved in the NewsReader project) with competencies in Natural Language Processing, Knowledge Representation, and Decision Support, were asked to contribute with some proposals of operations they would expect to use to interact with the KnowledgeStore.\footnote{Our previous experience in the LiveMemories project also contributed to provide some further operations.} To guide the collection of these operations, a template was provided asking for each of them: (i) a name; (ii) a description explaining the rationale of the operation; (iii) the input parameters; (iv) the expect output; (v) possible observations about the operation (e.g., further optional inputs, or variants); and, (vi) some usage examples.

Collected operations were then analysed to find commonalities, removing duplicates or operations subsumed by other ones, and investigating the possibility to further generalize them. Here below, we present some examples of the resulting operations, categorized according to the granularity of the information they are accessing: a single element of content, the content of a single layer, and the content of multiple layers.

**CRUD operations:** These operations refer to the possibility of creating (C), retrieving (R), updating (U) and deleting (D) a single element of content in the KnowledgeStore. CRUD operations are provided for each layer, while a couple of additional methods (storeResourceContent() and

<table>
<thead>
<tr>
<th>name</th>
<th>getMentionsByType()</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>lists all the mentions of a certain type, returning a selected set of attributes for each</td>
</tr>
<tr>
<td>input</td>
<td>type of mention and attributes to return</td>
</tr>
<tr>
<td>output</td>
<td>list with the mention identifier and the selected attributes for each matching mention</td>
</tr>
<tr>
<td>notes</td>
<td>none</td>
</tr>
<tr>
<td>example</td>
<td>return name and surname of PER mentions</td>
</tr>
</tbody>
</table>

**Inter-layer retrieval operations:** These operations refer to the possibility to retrieve some objects by exploiting content distributed over two or more layers of the KnowledgeStore; again, operations dealing with entities / statements may require some automated reasoning (e.g., to retrieve all the events about sports requires to consider also subsumed event types such as football events). As an example, we report the operation for retrieving all the mentions in the KnowledgeStore of a specific linguistic type (e.g., person):

<table>
<thead>
<tr>
<th>name</th>
<th>getResourcesFromEntity()</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>given an entity, return a list of news in which it is mentioned</td>
</tr>
<tr>
<td>input</td>
<td>the entity URI</td>
</tr>
<tr>
<td>output</td>
<td>a list with essential info for each matching resource (e.g., id, title and date)</td>
</tr>
<tr>
<td>notes</td>
<td>an optional parameter may determine the amount of info to be returned</td>
</tr>
<tr>
<td>example</td>
<td>get resources mentioning entity nwr:E105</td>
</tr>
</tbody>
</table>
It is worth noting that while CRUD operations can easily scale, intra- and inter-layer retrieval operations may be potentially affected by performance issues due to their greater complexity and the (possible) involvement of automated reasoning. To mitigate these issues, especially for operations requiring a “real-time” response, appropriate techniques need to be implemented (e.g., denormalization and additional indexes to avoid expensive joins), as reported in Section V-A.

V. THE KNOWLEDGESTORE ARCHITECTURE

The KnowledgeStore is a scalable, fault-tolerant storage server designed to be deployed on a cluster or in a cloud environment, and whose content can be accessed and manipulated by client applications through an API.

An high-level overview of the KnowledgeStore architecture is shown in Fig. 5. The server internal architecture is shown in the bottom part and is composed of three main components, described in the remainder of this section:

- HBase & Hadoop realize the primary storage for the structured and unstructured content of the data model, providing horizontal scalability and fault-tolerance;
- the Triple Store indexes a subset of statements to support inference and online semantic query answering, which cannot be easily and efficiently realized in HBase;
- the Frontend realizes the API of the KnowledgeStore, on top of the other components;

In addition, a number of tools are envisioned to support the management of the server infrastructure, the gathering of statistics, and the efficient bulk loading of unstructured and structured contents.

A. HBase & Hadoop

The primary storage for the KnowledgeStore content is provided by Hadoop\textsuperscript{14} and HBase\textsuperscript{15}, two open source frameworks developed by Apache and featuring distributed computation, replication and fault tolerance with respect to network and node failures. Hadoop provides a distributed (partitioned and replicated) file system and is used to store the unstructured contents of resources (e.g., news texts). HBase is a column oriented NoSQL database and is used to store the structured content of the KnowledgeStore. HBase is particularly suited for random, real time read / write access to huge quantities of data (such as big data); it provides a consistency model looser than traditional relational databases in order to achieve horizontal scalability and higher availability.

Three HBase tables are employed by the KnowledgeStore to store resources, mentions and statements, while entities are indirectly stored as the subjects of statements. Resources are indexed using a primary key derived from their most relevant attributes (e.g., publication date and source), in order to allow fast retrieval based on them. The primary key of mentions derives from the one of the containing resources, so to efficiently access all the mentions of a resource. Statements are indexed by their subject first, thus allowing a quick retrieval of the RDF description of an entity. Object relations, such as \texttt{dc:isPartOf}, \texttt{refersTo} and specialized relations in the configurable part of the data model are implemented with an association table, which is indexed by the relation type and the identifiers of the two related objects (to speed up retrieval, also inverse relations are stored); additional attributes characterizing a relation can be stored in this table.

Redundant tables and schema denormalization are employed to avoid expensive join operations when implementing the KnowledgeStore API. In particular, an auxiliary table allows to quickly navigate from an entity to its mentions and the containing resources (efficient navigation in the other direction is enabled by the choice of primary keys), while index tables specific to a KnowledgeStore instance can be defined at the resource and mention level to speed up retrieval of objects matching certain attributes. Denormalization is controlled by distinguishing between most-frequently accessed and least-frequently accessed attributes at every layer: most-frequently accessed attributes of an object are replicated in the records of referring objects, so to avoid additional join operations.

B. Triple Store

Statements are partially indexed in a triple store in order to enable efficient, inference-aware query answering. Indexing affects only those statements that satisfy certain configurable criteria; this allows, for instance, to exclude from inference statements whose extraction confidence level is below a given threshold. Each statement is stored as a \langle subject, predicate, object \rangle triple within a named graph \cite{DBLP:journals/tois/GeertsCS08} that encodes the context where the statement holds. Contextual statement metadata is associated to the graph via additional triples, and the graph is shared by all the statements holding in that context; additional statement metadata are not indexed. The SPARQL language \cite{SPARQL} is used to query and manipulate indexed data.

Logical inference aims at deriving the additional statements implied by stored data (ABox) and the ontologies defining its schema (TBox), and making them available as possible answers to applications and users queries. For instance, if a statement describes Volkswagen as a PublicCompany and PublicCompany is a subclass of Company in the TBox, then a query for all companies is expected to return Volkswagen as an answer. Inference in the KnowledgeStore must cope with

\footnotesize{\textsuperscript{14}http://hadoop.apache.org} \textsuperscript{15}http://hbase.apache.org
the large amount of data available as well as its contextual validity. The first aspect asks for a scalable and efficient approach, such as rule-based inference that scales to large amounts of data, combined with offline pre-materialization of the logical closure that speeds up online query answering. The second aspect asks for a custom logical language (implementable using inference rules) since no standardized ontological language currently supports inference on contextualized data. Both mentioned techniques—closure materialization and custom rule-based inference—are supported by triple stores\textsuperscript{16}.

Different triple store implementations offer different performance, scalability and fault-tolerance characteristics, as well as licenses. The use of the OpenRDF Sesame Java API permits different triple store implementations to be plugged in the KnowledgeStore. The Open Source Edition of the Virtuoso triple store\textsuperscript{17}, a product showing excellent performances in recent (April 2013) benchmarks\textsuperscript{18}, has been chosen for the current implementation of the KnowledgeStore. The Open Source Edition is limited to a single node deploy, where it can easily handle a billion of triples (as a reference, the English DBPedia consists of 400M triples). Additional scalability and transparent fault tolerance can be obtained using the (commercial) Enterprise Edition, or moving to open source clustered triple stores such as Bigdata\textsuperscript{19} or 4Store\textsuperscript{20}.

C. Frontend

The frontend component implements the external API of the KnowledgeStore by dispatching client requests to other components. It also controls the replication of data on auxiliary and denormalized tables in HBase as well as the indexing of statements in the triple store and the triggering of the necessary inference, which are transparently performed each time data is written through the API.

The majority of API operations is forwarded to a single component, either HBase & Hadoop or the triple store; they include all the CRUD and intra-layer operations, as well as several inter-layer operations. The remaining inter-layer operations are \textit{mixed queries} which can be decomposed into one or more semantic queries, targeted at the triple store, and one or more retrieval operation for structured and unstructured data in Hadoop or Hbase. An example is a query for news mentioning that “Barack Obama” participated to a sport event, which involves both a semantic query for events participated by Obama in the triple store, followed by a query on HBase for the documents mentioning those events. More complex queries are also possible, such as a query for all the managers mentioned in news about the “Lehman Brothers” bankruptcy, which requires an additional HBase lookup for person mentions in matching news (retrieved as before) and a semantic query to identify which of them are known to be managers. For all these mixed queries, it is a responsibility of the frontend to orchestrate their execution, by invoking internal components in the proper order, filtering and composing the final results.

Efficient communication protocols and compact data formats are crucial for the API implementation in the frontend. In particular, efficiency and ease of use for clients can be obtained using binary Remote Procedure Call (RPC) mechanisms such as Apache Thrift\textsuperscript{21} or Google Protocol Buffers\textsuperscript{22}, which allow for the automatic and cross-language generation of the client and server implementation of the API, based on an abstract specification written in an Interface Description Language (IDL). Streaming solutions and polling / notification mechanisms are deployed to deal, respectively, with the exchange of large quantities of data and with long running API operations. In order to enable the deploy over an insecure channel such as the Internet, basic security measures are provided consisting of username / password authentication of client applications and selective encryption of exchanged data that may contain sensitive or copyrighted information.

In order to avoid single points of failure, the Frontend component is replicated, taking inspiration from the approach utilized by the HBase and Hadoop daemons. Replication can be achieved by storing all the system state in HBase and deploying a suitable locking mechanism (or even a transaction manager such as OMID\textsuperscript{23}), to coordinate the access of multiple frontend instances to such state.

VI. RELATED WORK

The development of frameworks able to store integrated and interlinked unstructured and structured content has not been deeply explored in the literature.

Some investigations were carried out on document repositories based on semantics (e.g., [12], [13]). In these approaches, ontologies are used to represent domain vocabulary and the document structure, and they are used to annotate documents and document parts. However, the repository adopting these approaches (i) emphasise the document structure (e.g., tables, title) rather than document content, (ii) they do not foresee an integrated framework for storing semantic content and unstructured documents together, and (iii) they are typically not meant to be applied in big data contexts.

Relevant for our work is the contribution presented in [14]. The authors present a framework, based on a RDF triple store, that enables querying the bioinformatics scientific literature and structured resources at the same time, for evidence of genetic causes, such as drug targets and disease involvement. Differently from our approach, this work does not support multimedia content (triple stores currently provide only a limited support for integrating knowledge with unstructured resources, often consisting in simple full text search capabilities on RDF literals), and the framework is focused only on specific types of named entities appearing in the unstructured content, whereas

\textsuperscript{16}Rule-based inference is directly offered by some triple stores, and can be otherwise realized via fix-point evaluation of SPARQL queries.

\textsuperscript{17}http://virtuoso.openlinksw.com/dataspace/doc/day/wiki/Main/

\textsuperscript{18}http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/results/V7/

\textsuperscript{19}http://www.bigdata.com/bigdata/blog/

\textsuperscript{20}http://4store.org/

\textsuperscript{21}http://thrift.apache.org/

\textsuperscript{22}http://code.google.com/p/protobuf/

\textsuperscript{23}https://github.com/yahoo/omid/wiki
a rich, unconstrained set of entities and mentions can be managed in the KnowledgeStore.

Of some relevance are also frameworks for the extraction and storage of knowledge from Wikipedia, although they focus on providing a knowledge base and not an interlinked structured/unstructured knowledge source. For instance, in [15] the authors of YAGO 2 describe a framework to represent contextualized knowledge extracted from Wikipedia. Facts, mainly contained in Wikipedia Infoboxes, are enriched with three additional dimensions (time, location, context), and the whole content (447 million facts, 9.8 million entities) is stored in a relational database (PostgreSQL). However, this framework does not support the interlinking of an entity with the exact position within) the document where it is mentioned (the mention layer in our approach), as only the extracted information is stored.

Although exploited in a different context, dealing with much smaller quantity of content, also semantic desktop applications (e.g., MOSE [16], Nepomuk [17]) are partly related with the contribution here presented. Semantic desktop applications enrich documents archived on the personal PC of a user with annotations coming from ontologies. However, annotations are attached to the object associated to the document, and not to its content, thus not fully supporting the interlinking between unstructured and structured content.

VII. Conclusions and Future Work

In this paper we described the design of the KnowledgeStore, a framework enabling to jointly store, manage, retrieve, and semantically query, both unstructured and structured content. Thanks to the explicit representation and alignment of unstructured and structured information, the KnowledgeStore enables the development of enhanced applications (e.g., advanced decision support tools for professional decision-makers), and permits the design and empirical investigation of several information processing task (e.g., cross-document coreference resolution, knowledge fusion and crystallization).

The implementation of the KnowledgeStore according to the presented design criteria is currently in progress. The framework will be deployed in the NewsReader project, where it will be evaluated from a functional perspective based on its capability to (i) store an overwhelming24 daily stream of economical and financial contents (news articles and data), (ii) support a complex NLP pipeline in extracting knowledge from those contents, and (iii) provide suitable online and offline query capabilities for use in a decision support tool for professional decision-makers. In the same context, we also plan to carry out an extensive performance evaluation in terms of scalability with respect to data size, query load, and tolerance to nodes and network failures.

24A large international information broker, such as the NewsReader project partner LexisNexis (http://www.lexisnexis.com), typically handles about 2 million news each day, cumulating to an impressive 25 billion documents archive spanning several decades.

ACKNOWLEDGMENTS

The research leading to this paper was supported by the European Union’s 7th Framework Programme via the NewsReader Project (ICT-316404).

REFERENCES