Leveraging Flexible Data Management with Graph Databases

Elena Vasilyeva[1] Maik Thiele[2] Christof Bornhövd[3] Wolfgang Lehner[2]

[1]SAP AG
Dresden, Germany
elena.vasilyeva@sap.com

[2]Database Technology Group
Technische Universität Dresden, Germany
firstname.lastname@tu-dresden.de

[3]SAP Labs, LLC
Palo Alto, CA 94304, USA
christof.bornhoevd@sap.com

ABSTRACT

Integrating up-to-date information into databases from different heterogeneous data sources is still a time-consuming and mostly manual job that can only be accomplished by skilled experts. For this reason, enterprises often lack information regarding the current market situation, preventing a holistic view that is needed to conduct sound data analysis and market predictions. Ironically, the Web consists of a huge and growing number of valuable information from diverse organizations and data providers, such as the Linked Open Data cloud, common knowledge sources like Freebase, and social networks. One desirable usage scenario for this kind of data is its integration into a single database in order to apply data analytics. However, in today's business intelligence tools there is an evident lack of support for so-called situational or ad-hoc data integration. What we need is a system which 1) provides a flexible storage of heterogeneous information of different degrees of structure in an ad-hoc manner, and 2) supports mass data operations suited for data analytics. In this paper, we will provide our vision of such a system and describe an extension of the well-studied property graph model that allows to "integrate and analyze as you go" external data exposed in the RDF format in a seamless manner. The proposed integration approach extends the internal graph model with external data from the Linked Open Data cloud, which stores over 31 billion RDF triples (September 2011) from a variety of domains.

Categories and Subject Descriptors

H.2.1 [**Logical Design**]: Data models, Normal forms, Schema and subschema; H.2.5 [**Heterogeneous Databases**]: Data translation

General Terms

Algorithms, Design

Proceedings of the First International Workshop on Graph Data Management Experiences and Systems (GRADES 2013), June 23, 2013, New York, NY, USA.

Keywords

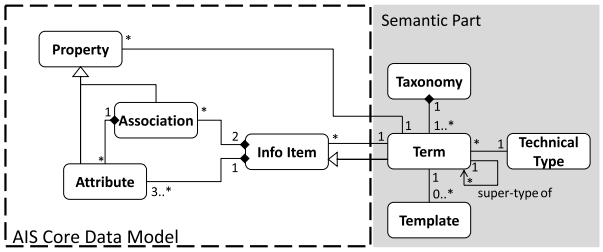
Graph Database, Linked Open Data, Property Graph

1. INTRODUCTION

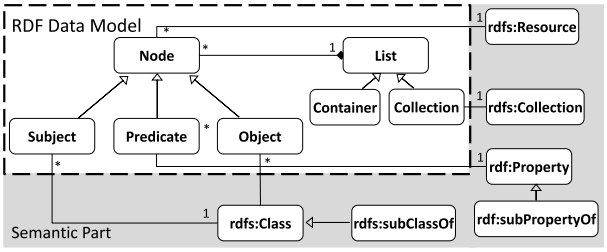
Data analytics and business intelligence have enjoyed immense popularity and success over the last ten years and now play a key role in corporate decision-making. However, due to the ubiquitous presence of data residing beyond the corporate boundaries, the requirements posed to data analytics have changed toward more agile and situational analytics. In contrast to conventional data warehouses approaches where all datasets are known at design time, situational analytics demands data provisioning, integration, transformation, and consolidation in an ad-hoc fashion. One popular example of such an external and very valuable data source is the Linked Open Data (LOD) cloud which offers billions of structured and irregularly structured information pieces. To make productive use of the variety of LOD, two things are required: first, a powerful schema-flexible data store and, second, a way to integrate and analyze external data to bring it into a new context, to mix it with other data sources, and to gain knowledge and insights from it.

The call for a timely integration of new data sources, however, confronts today's data models and architectures with a serious problem. Its integration is typically prevented by heterogeneous data formats and data of different structure and meaning. A traditional approach requires a global rigid schema considering all possible types and formats making the method too inflexible and cost-inefficient. Therefore, we propose an ad-hoc data integration engine on top of SAP's Active Information Store (AIS) that is able to augment the AIS data store by data from LOD in a seamless manner. In this way, we want to bridge the gap between the analytical world and LOD and want to show the value of LOD for business analytics in general.

The rest of this paper is organized as follows: we will start by presenting a use case in Section 2 that motivates the need for a flexible and extensible data model suited for ad-hoc data analytics. We then outline in Section 3.1 the core features of the SAP Active Information Store as well as the underlying data model that allows data integration in a "pay as you go" manner. Additionally, we briefly review the principles of the RDF data model in Section 3.2 and compare both models in Section 3.3. This comparison forms the basis of our architecture which is outlined in Section 4. Finally, we summarize our findings and point out directions for future work in Section 5.



(a) The AIS data model



(b) The RDF/RDFS data model

Figure 1: Graph data models

2. ENRICHMENT OF BUSINESS ANALYTICS WITH MARKET INSIGHTS

In the following, we want to give an example on how integration of external information can extend internal business analytics and enrich it with additional insights. Therefore, let us imagine a manager of a software company that recently launched a new mobile application that was widely advertised by the marketing department. After a good start and a lot of sales in the first month, the manager recognizes a steep decline in the second month. To analyze the reasons for that decline and to predict future sales, the manager has to conduct a what-if analysis. For this purpose, he requires a clear picture of the market state, information about promotions and the competing products as well as all additional data like user comments, sentiment information etc. Whereas a small part of this information is already available in corporate databases or is stored in well-defined and structured datasets, the majority has to be obtained from external sources, for example, the Linked Open Data cloud (see Section 3.2). This data must be provided during runtime in an ad-hoc manner leading to schema changes and modifications that cannot be anticipated during design time. In addition, due to the explorative nature of data analytics, the user should be able to stepwise integrate, mix, and analyze local and external data. As we can see, there are several issues to be solved in order to support users in performing data analysis in today's dynamic business environments. In detail, we need a platform that provides a flexible data model, supporting the co-existence of structured as well as irregularly structured information with a unified representation and access, allows ad-hoc queries to multiple sources and performs schema matching and integration between external and internal data models.

3. GRAPH DATA MODELS

In this section we present two underlying models that we use in our approach. These are the graph data model of the SAP HANA database, providing a flexible way of storing and using heterogeneous information, and the RDF-model of the publicly available Linked Open Data cloud.

3.1 The AIS Data Model

Modern business processes are characterized by an increasing number of participants and their heterogeneous data. To support such dynamic processes, we need a data model that is able to integrate information with varying degree of structure (structured, irregularly structured and unstructured data) and allowing its uniform handling. Such a flexible storage is introduced by the Active Information Store (AIS) [3], a schema-flexible graph data store provided by the SAP HANA database.

As it can be seen in Figure 1(a), the AIS data model consists of two parts: a core component and its semantic extension. The core component represents the data in the form of property graphs [17] constructed from vertices and edges between them. Vertices introduce entities, which are described with a number of attributes and corresponding values. Relationships between entities are drawn by directed edges together with their attributes.

Entities are represented by *Info Items*, which are basic processing units of storage, retrieval, extraction, and interrelation of data in the AIS data model. An Info Item has a set of *Properties* that are denoted in the form of *Attributes* or *Associations*. While an Attribute assigns a value to an Info Item, an Association is a unidirectional relationship between two Info Items and represents an edge in a graph. Properties are classified into three groups: mandatory (assigned to each Info Item), expected (available for most Info Items), and optional Properties (provided for some Info Items).

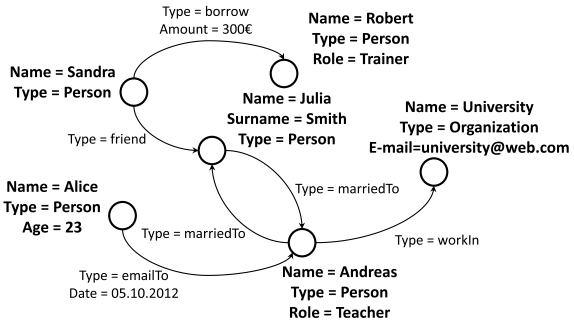


Figure 2: An example of a property graph

The example in Figure 2 shows a simple graph consisting of six nodes with different numbers of Properties. For example, the Info Item with the name "Sandra" has three Properties: its Name and two Associations. Associations bind Info Items in a graph and introduce relationships between them, for example, Julia married to Andreas.

The extension of the AIS data model describes a property graph semantically. A semantic type (*Term*) is assigned to each Info Item and its Properties. Terms are Info Items themselves that enables their processing as metadata or as actual data. Terms are connected with each other in a hierarchical *Taxonomy*, which describes a particular domain. The Terms can be obtained from domain-specific Taxonomies [3] and then be extended as needed.

The AIS data model presents a flexible data model in the form of an extended version of a property graph and sup-

ports the integration of data of different degrees of structure. It does not insist on the use of a common structure but it provides a way of co-usage of different information in a uniform way. The schema does not have to be determined at the beginning of a process, but it can easily evolve over time.

To allow flexible storing of heterogeneously structured information, the AIS data model supports the following key aspects:

- *Uniform information representation*: Different kinds of information can be presented in the AIS data model in the form of entities with attributes and relations between them. To distinguish between entities, objects' types (Terms) are introduced.
- *Schema flexibility*: The AIS data model does not enforce a static schema. The schema can evolve: new nodes, attributes, and associations can be added.
- *Support of continuous data integration*: Specific operators in the graph query and manipulation language WIPE [3] support stepwise data integration, give opportunities to change already stored information and to enhance it with additional characteristics.
- *Uniform handling of data and its metadata*: While the AIS data model distinguishes between semantic information (Terms) and actual data via their types, their handling is done in the same way. Therefore, they can be treated as metadata as well as actual data depending on the application at hand.

To query and manipulate data in AIS, we use the WIPE language [3], which offers standard operations like INSERT, UPDATE, DELETE, and LOAD at the level of Info Items and their Properties. In addition, WIPE supports graph traversal for resolving of Associations and Attributes.

In summary, the AIS provides a schema-flexible data store, which is additionally supported by powerful analytical functions of the SAP HANA database.

3.2 The RDF/RDFS Data Model

While an internal graph model is used only inside a company and its data is typically not available for a third party, Linked Open Data (LOD) is freely available in the Internet and can be legally reused by everybody. According to [2], "Linked data is a set of best practices for publishing and connecting structured data on the Web". These principles were applied to the LOD project that aims to make open data sets available for use. For this purpose, the data is converted into Resource Data Framework Format (RDF) and published on the web.

The Resource Description Framework [9] has been developed for describing metadata about web resources by representing properties and relationships between them. The relationships can be interpreted as graphs where nodes stand for web resources and properties and their relationships are drawn by edges. The basic structure of the RDF data model is a statement consisting of three parts: a subject, a predicate, and an object (see Figure 1(b)). While the first two parts have to be unique identifiers, an object can be a unique identifier or literal. Unique identifier (URI) is "a compact sequence of characters that identifies an abstract or physical resource" [12]. It combines a location of a resource and a name that remains unchanged over time. For example, the

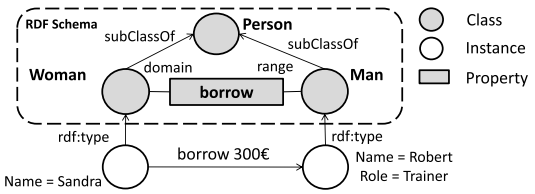


Figure 3: An example of a property graph with an RDFS extension

RDF statement "The car has color red" has a subject "the car", a predicate "has color", and an object "red".

RDF nodes without URI or literal, called blank nodes or anonymous nodes, are used to describe multivalued resources. For example, a person can have two living addresses consisting of a street and a house number. Therefore, each address can be described by a specific blank node. Multivalued objects can also be introduced by lists, which are classified in containers and collections. While containers are extensible, collections are closed lists. Due to the used order, collections can be of three types. A Bag represents an unordered set of elements, a Seq is an ordered set of elements, and an Alt provides alternatives of elements.

To express the example in Figure 2 in RDF, we have to create a set of statements for each Info Item and its Properties, and assign labels to the nodes. Then the Info Item, which Property "Name" has a value "Sandra", can be described by the following RDF Statements:

```
1: URI:personA URI:hasName Sandra
2: URI:personA URI:friend URI:personB
3: URI:personA URI:borrow URI:personC
4: URI:borrow URI:hasValue 300€
```

As we can see, to define one entity, we need multiple statements. Additionally, we have to create URIs that are unique inside an application. Moreover, to use them among nodes in the LOD cloud, we have to follow the general rules for publishing linked information.

To enable an entity construction and to establish classes and relationships, the RDF Schema (RDFS) [4] and OWL [7] can be used. While the RDFS is an extension of RDF, OWL is an extension of the RDFS. In Figure 1(b) the RDFS part is represented by the dark area outside the dashed box for the RDF data model. While a RDF statement shows a triple on an instance level, RDFS allows to go to an upper level and to describe which classes RDF parts relate to. The RDFS defines a data model for creation of RDF statements. With the RDFS we can define classes, relationships between them, their properties and build class and property hierarchies.

Figure 3 presents an RDFS-extension of the example from Figure 2. It is completed with a class structure: Sandra is of a class Woman and Robert is of a class Man, where Woman and Man are subclasses of a class Person.

RDF/RDFS data can be accessed via SPARQL [15], which supports queries among diverse sources of data in the RDF format. It allows querying of triple patterns, conjunction, disjunction, and optional patterns. Many data sets from the LOD cloud provide interfaces, called SPARQL endpoints, for querying their knowledge bases via the SPARQL language.

3.3 Comparing the AIS and RDF data models

Feature	AIS	RDF/RDFS
Graph type	Property graph	Labeled graph
Entity integrity	Guaranteed	No guarantee
Metadata	Special type	No strong separation from actual data
Technical types	Defined by semantic types	No data model support

Table 1: Comparison of graph data models

Although data can be converted easily between the AIS and RDF data models, there are several differences between them that have to be considered. While the AIS data model is an extended version of a property graph, the RDF graph is a labeled graph. The first one is a network of entities; the second one is a network of values. Therefore, we have to figure out which RDF statements relate to which entity and re-construct corresponding entities. In addition, multiple URIs can be assigned to the same entity. In this case an additional recognition step has to take place that cannot guarantee that all attributes and associations are discovered. For this purpose, synonym vocabularies can be used.

The RDF format does not distinguish precisely between meta information and actual data. Therefore, we have to use standards like OWL [7], RDFS [4], and application specific knowledge for correct storing data in the AIS data model.

In conclusion, the AIS data model expresses well an entity as a unit, supports technical types, and keeps meta information together with actual data, while the RDF/RDFS data model is very general and relies on a lot of conventions that prevent it from the usage as an internal business data model.

4. INTEGRATION APPROACH

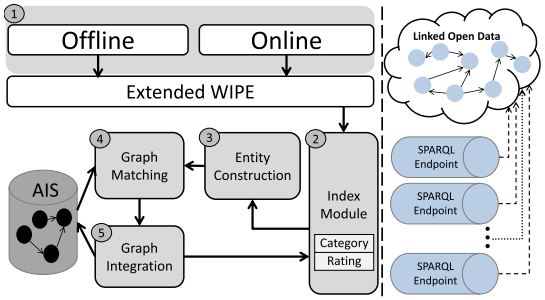


Figure 4: System architecture

To conduct graph extension in the AIS data model, we propose the creation of an intermediate data layer between the LOD cloud and the internal AIS data model (see Figure 4). This extension of the AIS data model can be integrated more tightly into the AIS engine in the future.

Triggering Graph Extension.

There are two ways to trigger graph extension with data from external sources: offline and online. An offline method means continuous graph extension. It does not depend on any incoming queries and it is not limited in time. The offline process is applied when general stable information is required for the analysis, e.g., geoinformation. Such data evolves slowly over time and typically stays unchanged during analysis.

By contrast to the offline procedure, the online process has a time threshold and an answer has to be provided to a user in realtime. Data to be integrated immediately can describe a current state of a market and can evolve over time dramatically. Therefore, this information has to be retrieved immediately and has to be kept only for the current analysis. An online method implies "on-the-fly" graph extension and is triggered explicitly by a user query "Extend a graph" or implicitly by the system. A query "Extend a graph" contains entities or their properties that have to be completed. The implicit online method can detect, whether the available information might not be sufficient to respond to a user's request: the answer has null values for some attributes or is empty. In this case, if a query starts with the "EXTEND" keyword, the online graph extension is launched.

Ad-hoc queries (online mode) for integration of external information are studied in [5]. The proposed system is able to process open-world queries: an extended version of PostgreSQL asks open data sources for the missing information. Although the presented solution is similar to our problem, it differs in the underlying model and, therefore, corresponding problems to be solved. While the authors use a RDBMS and open data sets, we focus on a graph data model and the LOD cloud, and on how to construct entities, compare them and to integrate then into a dynamic analytical process over graph data.

Some of the techniques proposed in this paper are similar to the methods presented in [18], where LDIF framework collects Linked Data from the web and put it into local representation. In comparison, our system works with two different models (RDF and AIS graph data model). In addition, we integrate not all data into an internal data model, but rather choose only relevant data and integrate it in an ad-hoc manner that requires user participation, query recomposition, and answer analysis.

Index Module.

To look for information, we have to decide, what, where and how deep to search. We introduce a single point of integration with external sources, the Index Module. It is responsible for the registration of SPARQL endpoints and the discovery of new information required for internal data analysis from the LOD cloud. Registration data is composed from addresses of SPARQL endpoints, their domains and ratings. Addresses and domain descriptions of these endpoints are available at system initialization and can be modified as needed. Ratings are derived from ratings of previously obtained results calculated by the Graph Matching Module, the number of triples of a corresponding triple store, and the source availability:

$$rating = rating_{average} + N_{triples}/N_{all} + A \qquad (1)$$

Each component in the equation has a value between 0 and 1. The first component is calculated by the Graph Matching Module as described later in the paper. The amount of data stored by a triple store is provided by its semantic description. The availability of a source for the last seven days can be aquired from the publicly available list of SPARQL

endpoints [1]. The Index Module chooses the SPARQL endpoint with the highest score for the rating function. If it is unavailable, the next endpoint on the ranked list is queried.

In [13] a strategy for selecting endpoints is studied. The presented approach heuristically selects only those endpoints that are able to process a triple pattern and provide relevant data. A given query is then decomposed into few subqueries, and each of them is sent to a corresponding endpoint. Although this approach enhances federated queries to the LOD cloud, relevant endpoints can be missed. In comparison, our approach chooses relevant sources based on the application domain and the context of a current query. In addition, endpoints are queried stepwise, which reduces the overhead of federated queries and guarantees the stepwise integration of only required information. Relevant sources are searched not only for querying but also for interlinking of a new repository with them. For this purpose, a semantic web index can be used [14], where a set of labels is extracted and used for keyword search over the LOD. Chosen sources are then refined with ontology matching methods. This approach presents a powerful search and classification of sources, but it is time-consuming, which makes it impossible to use for ad-hoc queries and incremental data integration.

In [10] the authors evaluate several strategies of source discovery and ranking. Some endpoints are known in advance to the system, others are discovered at runtime. The sources are ranked based on the triple pattern cardinality and specificity, join pattern cardinality, links to results, and retrieval costs. To apply it for our online method, it has to be adapted from triples to the corresponding entities, which requires cross-calculation of resource ratings.

Entity Construction.

After a SPARQL endpoint provided some results, we have to prepare the obtained triples for the internal data analysis using the Entity Construction Module. It is responsible for building entities from the LOD cloud. The component has to understand which subjects and objects are entities, which attributes they have and which relationships connect them. In other words, we have to distinguish between Info Items, their Attributes, and their Associations.

In the first step, we classify all triples into groups according to their subjects. All subjects with the same URI represent the same entity. In the second step, we create Properties: predicates and objects are stored as an Info Item's Properties. Finally, we have to determine, which Properties are Associations. The last step queries a SPARQL endpoint about its description and used RDFS. First, we construct queries with predicates corresponding to the RDFS, for example: predicates are "rdfs:Class" or "rdfs:Property". Then we substitute a subject or an object with an available URI from the Entity Construction Module.

After a SPARQL endpoint responded, we align the RDFS with the internal Taxonomy. The mapping can be done manually, automatically, or semiautomatically. In the first case, a user has to determine connections between Terms and Classes. In the second case, we propose to use synonym vocabularies. If a Term and a Class are synonyms then they potentially describe the same entity. Assuming we received a set of synonyms and there are two Classes that correspond to these synonyms. To choose the right one, we ask the user or we use the context information that is provided by the internal system for the mapping between the RDFS and the internal Taxonomy. All undiscovered Properties have to be processed as additional Attributes.

The discovered entities can be completed with more information from the LOD cloud. For this purpose, we can continue querying a SPARQL endpoint about all discovered URIs to the degree defined by a user. Then the process is done as in the first iteration with the original URI.

Graph Matching.

After the entities have been constructed, the Graph Matching Module starts to determine an optimal connection point for the data integration by matching constructed entities with entities from the internal graph. This component looks for connection points in the existing graph, whether there are same or similar entities. In our study we use the classification of schema-based matching techniques from [16],[19].

String-based similarity. String-based similarity is defined on the element level. It corresponds to the mapping between two entities on the level of Properties and their values. Properties are compared in isolation, no relationships are taken into account. They are considered as sequences of characters and can be compared based on their prefices, suffices, and the edit distance between them, or on calculated common n-grams [19]. Of course, several algorithms can be combined to increase the accuracy of the mapping.

Structure-level similarity. On this level we distinguish, whether entities have the same structure. The matcher considers a schema-level description without any instance data. We examine only mandatory Properties during matching.

Match cardinality. In addition to structural and string similarity, we consider the match cardinality [16]. The alignment between two Properties can have several mapping resolutions: the relationship can be one to zero, one to one, or one to many values. Then the integration of multivalued Properties leads to the extension of a Property's values.

Entity pairs are compared with an overall similarity measure, which comprises structural and string similarities with the weight of α and $(1 - \alpha)$, where α is an application-specific weight. This measure is combined from all URIs, which characterize an entity. After all entities are qualified, weighted ratings for the same URIs are summed up and sent to the Index Module for their incorporation into ratings of SPARQL endpoints (see Equation 1). Matching results typically contain not a single mapping, but a set of possible mappings between two graphs. To determine the final solution, the first matching result is used or a semi-automatic approach is applied where the decision is done by a user.

Graph Integration.

The last step, graph integration, is conducted by the Graph Integration Module. According to the mode of graph extension, online or offline, we distinguish between two ways to storing information. In the offline mode we propose storing data into the internal graph with additional information like a timestamp, URI, and the address of a contributing SPARQL endpoint. To inspect an evolving entity, we keep a sequence of its historical states. If a new value becomes available, we record it with time information and activate it. The size of a window for keeping values is user-specified.

In the online mode we keep data in the graph during the current analysis and provide linking for the future analysis. The linking means the use of pointers to a data source where the actual data can be obtained from. A linking Info

Item inherits Properties of its original entity. The Attribute values contain corresponding URIs with a timestamp.

A timestamp corresponds to a specific entity's state describing some changes. Therefore, to analyze the evolution of an entity, its states can be interlinked with each other with the help of a time decay and a clustering technique, like for example, in [11]. Although this method supports an entity's evolution and can be used for comparing with the current entity's state, it does not solve the problem of keeping an entity up to date. In comparison, in [6] constant monitoring of entities can be applied. Here, when a new twitter message arises, it is sent directly to the system. Our study focusses on the big LOD cloud; therefore, the constant monitoring of records will create intolerable overhead.

A scalable and novel solution is presented in [8], where new data is integrated during query processing. For this purpose, mapping rules are introduced at the schema level and at the instance level, which complete the data with additional information. The data is taken from local as well as remote sources of XML documents. Therefore, it does not take into consideration URI linking as proposed by the W3C Consortium or sophisticated construction of LOD entities.

5. CONCLUSION

In this paper we presented our project for enrichment of internal business data with heterogeneous external information provided in the RDF format from the LOD cloud. We presented 1) a system, which provides a schema-flexible data store of differently structured information and analytical capabilities of the SAP HANA database and proposed 2) an ad-hoc approach for the integration of external information into an internal graph. The system supports the offline extension of an internal graph as well as ad-hoc queries with missing information. The integration approach provides incremental and flexible data integration. The schema does not have to be defined at the beginning of the analysis, but rather evolves over time. After the decision to integrate new data has been taken, relevant external sources are queried. Then entities are created from obtained RDF triples and integrated into the internal graph by storing of a corresponding URI or values with time and source information. In detail, our work is distinguished by three key contributions. The first is the novel method for choosing relevant SPARQL endpoints based on their domain description and ratings of previous results. The second is the entity construction strategy that aligns external RDF information with the internal property graph. The third contribution is the integration strategy allowing direct access to a SPARQL endpoint for querying of up-to-date information. Future work is planned to implement the proposed solution and evaluate it using several business-related data sets.

6. ACKNOWLEDGMENT

This work has been supported by the FP7 EU projects ROBUST (grant agreement no. 257859) and LinkedDesign (grant agreement no. 284613).

7. REFERENCES

[1] SPARQL Endpoints Status. http://labs.mondeca.com/sparqlEndpointsStatus/.

[2] C. Bizer, T. Heath, and T. Berners-Lee. Linked data-the story so far. *IJSWIS*, 5(3):1–22, 2009.

[3] C. Bornhövd, R. Kubis, W. Lehner, H. Voigt, and H. Werner. Flexible Information Management, Exploration and Analysis in SAP HANA. In *DATA*, pages 15–28, 2012.

[4] D. Brickley and G. R.V. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, February 2004.

[5] J. Eberius, M. Thiele, K. Braunschweig, and W. Lehner. DrillBeyond: enabling business analysts to explore the web of open data. *Proceedings of the VLDB Endowment*, 5(12):1978–1981, 2012.

[6] M. Grinev, M. Grineva, M. Hentschel, and D. Kossmann. Analytics for the real-time web. *Proceedings of the VLDB Endowment*, 4(12):1391–1394, 2011.

[7] W. O. W. Group. OWL 2 Web Ontology Language. W3C Recommendation, December 2012.

[8] M. Hentschel, L. Haas, and R. J. Miller. Just-in-time data integration in action. *Proceedings of the VLDB Endowment*, 3(1-2):1621–1624, 2010.

[9] G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, February 2004.

[10] G. Ladwig and T. Tran. Linked Data Query Processing Strategies. In *The Semantic Web – ISWC 2010*, volume 6496 of *Lecture Notes in Computer Science*, pages 453–469. Springer, 2010.

[11] P. Li, X. Dong, A. Maurino, and D. Srivastava. Linking temporal records. *Proceedings of the VLDB Endowment*, 4(11):956–967, 2011.

[12] L. Masinter, T. Berners-Lee, and R. T. Fielding. Uniform resource identifier (URI): Generic syntax. 2005.

[13] G. Montoya, M.-E. Vidal, and M. Acosta. A Heuristic-Based Approach for Planning Federated SPARQL Queries. In *Proceedings of the 3rd International Workshop on Consuming Linked Data (COLD2012)*, 2012.

[14] A. Nikolov and M. d'Aquin. Identifying relevant sources for data linking using a semantic web index. In *Proceedings of the 4th Workshop on Linked Data on the Web (LDOW 2011)*, 2011.

[15] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, January 2008.

[16] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[17] M. A. Rodriguez and P. Neubauer. Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41, 2010.

[18] A. Schultz, A. Matteini, R. Isele, P. N. Mendes, C. Bizer, and C. Becker. LDIF-A Framework for Large-Scale Linked Data Integration. In *21st International World Wide Web Conference (WWW 2012), Developers Track, Lyon, France*, 2012.

[19] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. In *Journal on Data Semantics IV*, pages 146–171. Springer, 2005.