bacon: Linked Data Integration based on the RDF Data Cube Vocabulary

Sebastian Bayerl, Michael Granitzer
Media Computer Science
University of Passau
{forename.surname}@uni-passau.de

ABSTRACT

Discovering and integrating relevant real-live datasets are essential tasks, when it comes to handling Linked Data. Similar to Data Warehousing approaches, Linked Data can be prepared to enable sophisticated data analysis. The developed open source framework *bacon* enables interactive and crowed-sourced Data Integration on Linked Data (Linked Data Integration), utilizing the RDF Data Cube Vocabulary and the semantic properties of Linked Open Data. Discovering suitable datasets on-the-fly in local or remote repositories sets up the ensuing integration process. Based on well-known Data Warehousing processes, the semantic nature of the data is taken into account to handle and merge RDF Data Cubes. To do so, structure and content of the cubes must be analyzed and processed. A similarity measure has been developed to find similarly structured cubes. The user is offered a graphical interface, where he can search for suitable cubes and modify their structure based on semantic properties. This process is fostered by a set of automated suggestions to support inexperienced users and also domain experts.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Data warehouse and repository*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*

General Terms

Algorithms, Experimentation

Keywords

Linked Data, Data Discovery, Data Integration, Data Fusion

1. INTRODUCTION

By now, a great number of interlinked datasets are available in the Linked Open Data Cloud (LOD Cloud). When it comes to statistical data, huge datasets can be found. For example, statistics provided by eurostat[1] have been published as Linked Data containing some 8 billion triples[2]. Discovering relevant information that is potentially distributed over multiple remote repositories is not an easy task and still a relevant research topic. The intended purpose of the prototype framework *bacon* is to carry a workflow into effect that is composed of Data Discovery and Data Integration steps. It hereby specializes on statistical data that is published using the RDF Data Cube Vocabulary [4]. There is work, providing an overview over every dataset using this vocabulary in the LOD Cloud [9]. The sources of the bacon research prototype are released under the MIT license and can be found in a github repository[3]. The used vocabulary defines a multidimensional data structure, derived from the traditional OLAP (online analytical processing) cube, known from Data-Warehousing processes. Such datasets are herein after referred to as cubes.

Enabling further data analysis and finding new insights by integrating multiple isolated datasets is the motivation for Linked Data Integration. Therefore, the Data Integration process must be aware of the properties of Linked Data and their impacts, like the graph-based structure or the usage of URIs.

The prototype enables the fusion of semantically associated cubes and offers functionality for discovering related cubes, modifying their basic structure and integrating them into a single cube. In this regard, a set of problems can be inferred that must be tackled, considering the characteristics of Linked Open Data:

1. Reusing the RDF Data Cube Vocabulary, how can the vocabulary be extended to fit Data Integration specific needs like provenance information?
2. How can suitable cubes be discovered? Therefore, structure and content of the data must be considered to define a similarity measure for cubes.
3. How can an efficient Data Integration on Linked Data be conducted? Applying structural modifications and detecting duplicate information is part of this problem.

This paper is structured as follows: After the introduction, the problem settings will be discussed in detail in section 2. This work is based on the RDF Data Cube Vocabulary and therefore, section 3 will explain the purpose of this

[1]`http://ec.europa.eu/eurostat`
[2]`http://eurostat.linked-statistics.org/`
[3]`https://github.com/bayerls/bacon`

vocabulary and describe the necessary structural enhancements and adaptions. Following this, the actual approach (section 4) will be presented. The evaluation section 5 judges the performance and robustness of the current implementation. Section 6 will give an overview over the related work, considering Data Integration on relational and linked data. Finally, the findings are in section 7.

2. PROBLEM SETTING

Interlinking data is the last step of the deployment schema for Open Data[4]. Schema and data can be interlinked to provide additional context information fostering interoperability. Here, Linked Data Integration means that datasets are not only interlinked but the data is merged into a single dataset. This work is limited to the integration of statistical data that is structured in a multi-dimensional data model. To be able to work with homogeneous datasets, only cubes compliant to the RDF Data Cube Vocabulary are considered. An essential part of the problem setting is the discoverability of the cubes. In a basic use-case, the user is looking for suitable cubes in a local repository. For that purpose, the cubes must be identified as such and also a metric must be defined to measure suitability. This approach is not limited to the local repository, but can be applied to the LOD Cloud to discover cubes stored in remote endpoints. Relevant but unknown information can now be discovered, by making the cubes comparable. Given a set of cubes, compliant to the descriptions in section 3, the core problem setting is how to merge these graph-based datasets into a single homogeneous cube. This can enable further analysis or visualizations, what is not part of the problem setting in this context. The integration task is similar to the Data Integration step in the Data Warehousing process. In general, similar problems have to be solved but in respect to different data structures. At first, a target schema must be generated. Following, the data must be loaded from different sources and transformed into the target format. While integrating the data, key integrity must be maintained, while conflicts like duplicates must be handled. To be able to apply the merging process to cubes using the RDF Data Cube Vocabulary, the following essential problems must be tackled:

1. Modify the structure of the input cubes to fit the structure of the resulting cube (Integration on schema level).
2. Detect and handle duplicate observations and therefore maintain the global key integrity. This makes it necessary to develop an update strategy (Integration on instance level).

While corresponding Data-Warehousing algorithms can be reused in the context of merging Linked Data, the nature of Linked Data causes additional research questions, e.g. the equality of values cannot be simply inferred by comparing literal but e.g. same-as relationships of resources must be considered.

3. CUBE VOCABULARY

The primary use of the RDF Data Cube Vocabulary [4] is the sharing and publishing of multi-dimensional data using the W3C Resource Description Framework [5]. This vocabulary provides a structure to describe numerical facts and dimensions, similar to the OLAP cubes, known from Data-Warehousing systems. Using this W3C Recommendation brings advantages besides providing an open standardized data format. This implementation utilizes a subset of the available concepts from the vocabulary. An exemplary graphical representation of a complete cube can be found within the github project[5].

Basically, a valid data cube, which is a multi-dimensional data structure, consists of a dataset structure definition and a set of observations compliant to the structure. The structure definition describes which dimensions and facts are available using labels and URIs. For example the number of sales can be stored together with the dimensions, location, time and product. An observation is a data instance, applicable to the structure definition. Such a cube is therefore the Linked Data equivalent to the schema definitions and the data rows in a traditional Data Warehouse.

Originally, the *rdfs:range* definition of a dimension property can point to an arbitrary concept or literal type. To normalize this behavior, the generic concept entity was introduced. An entity then always provides a *rdfs:label* and additionally the disambiguated concept for this resource. Therefore, it is possible not only to provide the dimension value as a label, but also to name an optional disambiguated resource for the label.

Additional properties are introduced to normalize the set of metadata stored with the cube. The RDF Data Cube Vocabulary does not necessarily mention these properties but they can be subsumed as metadata. Therefore, information specific to the merging process can be stored with the cubes. To model the provenance informations, concepts from the W3C Recommendation: The PROV Ontology [7] are reused with the prefix *prov.*

4. APPROACH

In section 2 a set of problems have been described, which must be tackled to be able to implement a sophisticated Linked Data Integration prototype. This section will explain, how these tasks are solved and implemented in the research prototype.

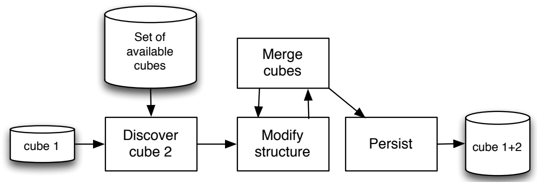


Figure 1: Merge two cubes with similar structure

For a quick overview, the essential high level workflow is depicted in figure 1. After selecting a first cube, the user can select a second cube from the set of already discovered cubes. Discovering unknown cubes is part of a separate workflow. The list of cubes is ordered by the similarity to the input cube. This similarity is computed by comparing the dataset structure definitions of the cubes. This similarity function, used for ranking the cubes, is described in detail in subsection 4.1. In general, cubes are merged by identifying their common structure and integrating their data into the target

[4]http://5stardata.info/

[5]https://raw.githubusercontent.com/bayerls/bacon/master/cube.jpg

schema, what is described in subsection 4.2. The user can optionally modify the structure of the merged cube. Subsection 4.3 shows the usefulness and functionality of these modifications. Like shown in figure 1, modifying the structure and merging the cubes is realized as a loop, to enable iterative modifications. Beside this user-driven workflow, other approaches have been implemented. Subsection 4.4 summarizes the major workflows of the framework.

4.1 Ranking cubes

Assuming a first cube was already selected, the prototype is capable of suggesting suitable candidate cubes. For this purpose, the cubes are ranked based on their structural similarity to the preselected cube. A higher number of common components indicates a higher similarity. In the basic implementation components are considered common, if equal resources are used. Therefore, it is sufficient to compare the dataset structure definitions of the cubes to compute their similarity. Depending on the overlap degree of the components a score between zero and one is computed. A score of zero denotes no match, whereby one denotes identical dataset structure definitions. Given two cubes c_1 and c_2, following values must be computed:

- d_1 (m_1) is the total number of dimensions (measures) in the first cube.
- d_2 and m_2 are defined respectively for the second cube.
- d_c (m_c) is the number of common dimensions (measures) in both cubes.

$$sim_{full} = (\frac{\frac{d_c}{d_1} + \frac{d_c}{d_2} + \frac{m_c}{m_1} + \frac{m_c}{m_2}}{4}) \tag{1}$$

$$sim_{dim} = (\frac{\frac{d_c}{d_1} + \frac{d_c}{d_2}}{2}) \tag{2}$$

Depending on these values, the similarity sim_{full} can be computed as shown in equation 1. For the sake of semantics, only components of the same type are compared. This similarity is suitable, if the user tries to find cubes with similar or equal components to extend the number of observations. Alternatively, the user might want to extend the number of measures in a cube. In this case, it is more suitable to compare only the dimensions of the cubes (Compare equation 2). If there are multiple equal scores, the timestamp is used as a second sorting criterion. More complex sorting techniques can utilize Linked Data properties. If different resources are used for the components, equality can be inferred if a path of same as relationships can be found between them. Functional dependencies can be found by exploring the subclass dependencies of the resources.

4.2 Merging cubes

This step is the core of the whole merging process and can be split into the following sub-steps.

1. Identify the common structure of the cubes to detect the target schema.
2. Generate the new cube with the identified structure and add the observations with respect to the new structure.
3. Guarantee that the key integrity constraint is not violated. No two observations must have the same values for the corresponding dimensions.
4. Add metadata to the cube.

The first task primarily falls back to identify common components in the dataset structure definition, which are either dimensions or measures. The assumed precondition is that the components are labeled and disambiguated with URIs, e.g., from the Linked Data Cloud. Only if syntactically equal concepts are used in both cubes, the component is considered as a part of the resulting cube. Finding non-trivial semantic equalities are explained in subsection 4.3.

Subsequent, the merged cube is generated with the target schema and the observations are added. Here, only the components of the observations are carried over that are actually part of the target schema. If both cubes contain observations with the same values for the dimensions, a key integrity violation is found. This conflict can be resolved, by applying predefined Data Fusion rules. Depending on the data at hand, different update strategies are reasonable. Per default, the values of the second cubes are carried over what can be interpreted as an update of the values of the first cube.

Finally, add the metadata to the merged cube, like provenance information, the current timestamp and the provided label and description. The prototype keeps track about the provenance of the merged cube. The source cubes are linked with the *wasDerivedFrom* property provided by the PROV-O ontology. The provenance information about the authenticated user is stored with the cube.

These steps do not require any interaction with the user. This automation enables additional workflows (See subsection 4.4).

4.3 Structure modifications

The merging is described in subsection 4.2 as an atomic and fully automated processing step. To join this with the structure modification functionality, these steps are arranged as a loop. The optional structure modification is always followed by a merging step. The computed result, in particular information about the occurrence of updates is presented to the user and can be used as a decision-support for the next structure modification step. Two atomic modifications can be applied, whereby the third one is a combination of the first two:

1. Add a component to a source cube with a default value.
2. Merge two unmatched components of the source cubes.
3. Add a component to the merged cube. This correlates to adding components to the source cubes and then merging them.

In the following, a simplified example will show the applicability of these modifications. Suppose there are two cubes containing the population figures over multiple years for different cities. There is a single dimension *year* and a single measure *population.* Merging such two cubes will probably result in unwanted updates, because both cubes can contain data for the year 2014. The information about the city is probably present in the description of the cubes but is missing as the distinguishing information in the dimension. Adding a distinguishing dimension *city* to the merged cube with different values for the source cubes, e.g., Berlin and Munich, can resolve this conflict. The value combination "Berlin - 2014" can now be distinguished from "Munich - 2014". Allowing more fine-grained modifications enables the handling of more situations. If the first cube already has the missing dimension *city* in contrary to the second cube, it then must be manually added and merged. Also the dimensions might be already present but using different resources for identification. The merge step will not recognize them as equal but the user can manually match these two components.

A strength of Linked Data is used by suggesting a list of potential merging candidates to the user. During the structure modification step, the user can query a service, which tries to discover same-as relationship between the component resources. If one is found, the equality of the two resources can be inferred and the components can be merged. In this case, the balloon service [14] is used for this purpose.

Structure modifications make it necessary to allow the user to choose new labels for the dimensions or measures. A disambiguation service [16] is integrated to assist the user in finding appropriate resources for these labels. On request, a list of concepts for the disambiguated label is presented to the user.

4.4 Workflows

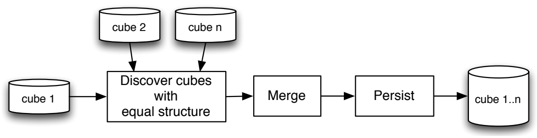


Figure 2: Merge large set of cubes (equal structure)

The similarity function for cubes provides additional areas of application. In combination with a crawler, it can be used to find mergable cubes in the LOD Cloud. Hereby it is not relevant, if similar cubes are stored in the same endpoint. Also cubes can be sorted and clustered for further managing and recommendation purposes. The prototype is implemented as a web application to enable user interactions with the graphical components, which guide the user through the main workflow of the prototype. All functionality, like the similarity functions, the structural modifications and the update-log visualization are integrated in the prototype. Also the external services, balloon and disambiguation, can be called via graphical components. Apart of the main workflow, a headless workflow has been implemented, which is depicted in figure 2. Hereby, merging without user interaction is possible by making some assumptions to enable a default behavior. The user only has to provide an initial cube to start the merging process. Suitable cubes to merge are found by selecting all local cubes with a perfect structural similarity based on all their components. Also, overlapping observations are treated as intended updates. This approach is suitable, if a large set of cubes with equal structure definitions are available. All cubes are automatically merged and persisted in the local triple store.

5. EVALUATION

This section will evaluate the features of the prototype. Therefore, different use cases are conducted to check functionality and performance of the components. The following itemization summarizes the test cases.

- Use all provided functionality to merge three datasets via the GUI. The ranked list is used to find the merge partners. Add dimensions and merge them before the final cube is persisted. (Subsection 5.1)
- Evaluate the performance of the ranking functions. (Subsection 5.2)
- Headless merge: Merge 25 cubes without user interaction. (Subsection 5.3)

The evaluation is performed on a MacBook Pro Intel Core i7 with 2,8GHz. The prototype and the attached triple store (blazegraph[6]) both run on a tomcat 7 in a Java 7 virtual machine and have assigned two gigabytes of heap size in each case. The prototype sends frequent queries to the triple store and therefore, it is deployed locally to minimize the impact of the network latency.

5.1 Manual approach

To ensure the functioning of the merger backend and the graphical user interface, a series of steps were developed, which cover every feature. If every step can be performed without failure and the resulting cube is valid, a functioning of the basic features can be inferred. The use case consists of the following steps:

1. With a preselected first cube, find and select the second cube using the ranked list of suggestions (Similarity = 1.0).
2. Every component must be merged and shown in the resulting structure.
3. The key integrity diagram must show an observation overlap.
4. Add dimensions to the cubes with explicit values.
5. Merge the unmatched dimensions.
6. The merged dimension now must be part of the resulting structure and the observation overlap must be resolved.
7. Store the merged cube with a new metadata.

Eurostat is the statistical office of the European Union and provides datasets with statistical data at European level. A self developed framework is used to lift some of these datasets into the cube format. These genuine cubes are now used for testing purposes. In particular two cubes about the quantity of specific fruit trees in the European countries are selected. They have an equal dataset structure definition containing the dimensions *location* and *year* together with the measure *quantity*. As expected an observation overlap was indicated due to the missing dimension *fruit*. The corresponding information was only available via the labels of the cubes. Manually adding the dimension with the values orange and apple resolved this conflict. The merge process was passed successfully, what indicates that the prototype meets the requirements.

[6]http://www.systap.com/blazegraph

5.2 Performance evaluation: ranking function

The triple store maintains an index for faster querying and the total number of triples influences its performance. To remove this influence the full set of data cubes is imported before the performance tests are started. In total, there are 3750 cubes, whereby the different test sets are distinguished by five unique provenance IDs. Therefore the total set can be divided into subsets with 250, 500, 750, 1000 and 1250 cubes. All of these cubes have the same dataset structure definition, which consists of 20 dimensions and ten measures. Here, the structure-based ranking functions show always worst-case performance, because no early structural distinction can be found. To evaluate the performance of the ranking functions, one cube of a set is compared with every cube of this set. Every test is repeated ten times and three unmeasured runs are executed in the beginning, to reduce undesired influences of the operating system. Figure 3 shows the performance of the two ranking functions.

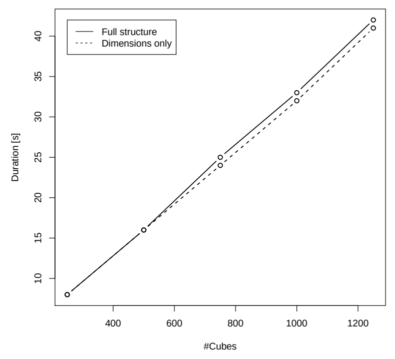
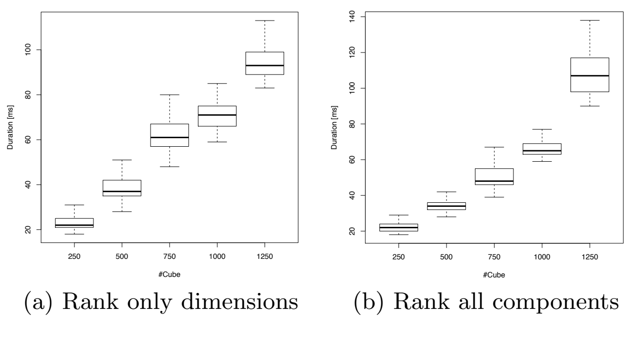


Figure 3: Ranking function evaluation (no cache)

Figures 3 depicts a linear dependency between time and the number of considered cubes. Further research showed that this runtime is not mainly influenced by the computation of the scores, but by querying the triple store. Implementing a simple cache mechanism showed a drastic performance improvement. Figures 4 (a) and (b) show the performance of the ranking function if the dataset structure definitions are hold in the main memory during the evaluation. Using very simple caching mechanisms for parts of the cube is unproblematic, because there are no possibilities to alter an existing cube at the moment.

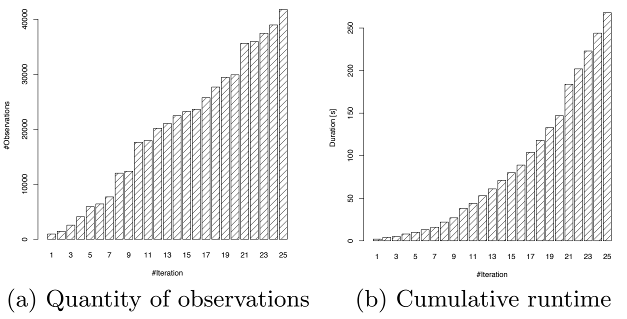


(a) Rank only dimensions (b) Rank all components

Figure 4: Ranking function evaluation (with cache)

5.3 Evaluation: headless merge

The evaluation of the headless mode was performed with 25 real world cubes, which were again downloaded from Eurostat and contain information about fishery landings in the European Union over multiple years. Every cube contains the data for a single country, whereby every cube has three dimensions (year, country and type of fish) and one measure (total landings in Euro). Every single cube contains multiple thousands of observations whereby the total sum of observations is 41776. The cubes are equally structured and every observation is uniquely identified.



(a) Quantity of observations (b) Cumulative runtime

Figure 5: Integrating 25 cubes (headless)

Figure 5(a) depicts the number of observations after every merge iteration. This information is relevant to be able to interpret figure 5(b), which depicts the accumulative time consumption for the merge iterations. The figures show, that the number of observations per cube has a huge influence on the merging process and that the time for merging doesn't scale linearly with the number of observations. Here again, the performance might be again highly dependent on the performance of the triple store, similar to the findings in subsection 5.2. Not storing every intermediate cube in the triple store, together with a caching mechanism is a possible approach to improve the performance. Hence, this is a functional test it has been shown that it is able to merge multiple cubes without human interaction and persist a merged cube.

6. RELATED WORK

Traditional Data-Warehouses offer sophisticated methods to create, maintain and query these databases in an efficient way. These methods are outlined in [3]. Relevant methods are Data-Cleaning and Data-Transformation, which are part of the Data-Integration process. Data-Cleaning deals with tasks like, migrating, scrubbing and auditing data. These steps prepare the data for the actual integration process. The Data-Transformation process adapts the data to the schema of the database.

The Linked Data Integration Framework (LIDF)[15] is able to crawl Linked Data from the web and uses mappings to translate the data into a local target vocabulary. In this context, vocabulary heterogeneity, URI aliases and the quality of the data are tackled as major problems. To integrate the different vocabularies into a single target schema, a set of mappings must be formulated. Silk[6] is used to identify resources, which describe the same real-world entity and Sieve[8] is used for Data Quality Assessment and Fusion.

Sieve was developed as a module of LDIF. Data Fusion is hereby defined as the integration of duplicate data rows, which are possibly inconsistent into a single and clean representation [1, 2]. Sieve fuses data using quality scores based on user-configurable quality assessment policies during the import phase of the Linked Data.

The ODCleanStore (ODCS) Framework[10] contains a module the enables Data Fusion during query execution. Deciding and mediating conflict resolution policies are deployed depending on the predicates of the relevant triples.

The KnoFuss system [13] focuses on instance-level integration of Linked Data. It uses the Jaro-Winkler distance or a clustering algorithm using TF-IDF and N-grams metrics to find conflicting resources.

The authors of [11] propose a semi-automatic method for the identification and extraction of valid facts form semantic data repositories. They also propose a multi-dimensional data structure to store the data using the resource description framework. A similar approach can be found in [12].

These approaches have map (semi-) structured sources to a single target schema per use-case or describe a part of this process. Data is optionally lifted and transformed in respect to predefined rules, whereby the Data Fusion techniques completes the ETL process. *bacon* operates on already lifted data and focuses on a more dynamic aspect of Data Integration. The target schema and the relevant datasets are not necessarily certain afore, but can be modified during the integration process. This allows the user to react interactively to different schemas of the datasets.

7. CONCLUSION & FUTURE WORK

This paper presented a Linked Data Integration prototype specifically tailored for the RDF Data Cube Vocabulary. It enables a basic integration functionally and allows the sorting of cubes and a dynamic modification of the cubes' structure during the merging process. There are multiple components implemented to exploit the properties of Linked Data. The equality of components is interred by using same-as relationships, whereby a disambiguation service finds suitable resources for labels. The graphical user interface guides the user through every step of the process and is enriched with a set of components, which suggest possible steps and additional information. Hereby a coherent experience for non-expert users is created. The evaluation showed the functioning of the prototype in different scenarios. It is therefore an enabler for further data analysis or possible visualizations. The evaluation also showed a high dependency on the underlying triple store. Future work will contain runtime improvements. Different triple stores will be evaluated to compare their runtime behavior. Also a full-fledged caching system for the dataset structure definitions is thinkable due to their small size. This can also be applied to the observations. Depending on their quantity, in-memory merging or a batch processing can be implemented.

8. ACKNOWLEDGMENTS

The presented work was developed within the EEXCESS project funded by the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement number 600601.

9. REFERENCES

[1] J. Bleiholder, K. Draba, and F. Naumann. FuSem: Exploring Different Semantics of Data Fusion. *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 1350–1353, 2007.

[2] J. Bleiholder and F. Naumann. Data Fusion. *ACM Computing Surveys*, 41(1):1–41, Dec. 2008.

[3] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1):65–74, Mar. 1997.

[4] R. Cyganiak and D. Reynolds. *The RDF Data Cube Vocabulary.* W3C Recommendation. http://www.w3.org/TR/vocab-data-cube/.

[5] R. Cyganiak, D. Wood, and M. Lanthaler. *RDF 1.1 Concepts and Abstract Syntax.* W3C Recommendation, 2014. http://www.w3.org/TR/rdf11-concepts/.

[6] R. Isele, A. Jentzsch, and C. Bizer. Silk Server - Adding missing Links while consuming Linked Data. *1st International Workshop on Consuming Linked Data*, 2010.

[7] T. Lebo, S. Sahoo, and D. McGuiness. *PROV-O: The PROV Ontology.* W3C Recommendation, 2013. http://www.w3.org/TR/prov-o/.

[8] P. N. Mendes, H. Mühleisen, and C. Bizer. Sieve: Linked Data Quality Assessment and Fusion. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 116–123, New York, New York, USA, 2012. ACM.

[9] A. Meroño-Peñuela. LSD dimensions: Use and reuse of linked statistical data. In *Knowledge Engineering and Knowledge Management - EKAW 2014 Satellite Events, VISUAL, EKM1, and ARCOE-Logic, Linköping, Sweden, November 24-28, 2014. Revised Selected Papers.*, pages 159–163, 2014.

[10] J. Michelfeit and T. Knap. Linked Data Fusion in ODCleanStore. *International Semantic Web Conference (Posters & Demos)*, pages 1–4, 2012.

[11] V. Nebot and R. Berlanga. Building Data Warehouses with Semantic Data. In *EDBT/ICDT Workshops*, New York, New York, USA, 2010. ACM Press.

[12] M. Niinimäki and T. Niemi. An ETL Process for OLAP Using RDF/OWL Ontologies. *Journal on Data Semantics XIII*, pages 97–119, 2009.

[13] A. Nikolov, V. Uren, and E. Motta. Towards Data Fusion in a Multi-ontology Environment. *Proceedings of the WWW2009 Workshop on Linked Data on the Web*, 2009.

[14] K. Schlegel, F. Stegmaier, S. Bayerl, M. Granitzer, and H. Kosch. Balloon fusion: Sparql rewriting based on unified co-reference information. In *5th International Workshop on Data Engineering Meets the Semantic Web, co-located with the 30th IEEE International Conference on Data Engineering*, 2014.

[15] A. Schultz, A. Matteini, R. Isele, P. Mendes, C. Bizer, and C. Becker. LDIF - A Framework for Large-Scale Linked Data Integration. *21st International World Wide Web Conference*, pages 1–3, 2012.

[16] S. Zwicklbauer, C. Seifert, and M. Granitzer. Do we need entity-centric knowledge bases for entity disambiguation? In *Proceedings of the 13th International Conference on Knowledge Management and Knowledge Technologies*, i-Know '13, pages 4:1–4:8, New York, NY, USA, 2013. ACM.