Linked Open Data Aggregation: Conflict Resolution and Aggregate Quality

Tomáš Knap, Jan Michelfeit, Martin Nečaský
Department of Software Engineering
Charles University in Prague
Prague, Czech Republic
Email: tomas.knap@mff.cuni.cz, michelfeit.jan@gmail.com, martin.necasky@mff.cuni.cz

Abstract—The paradigm of publishing governmental data is shifting from data trapped in relational databases, scanned images, or PDF files to open data, or even linked open data, bringing the information consumers (citizens, companies) unrestricted access to the data and enabling an agile information aggregation, which has up to now not been possible. Such information aggregation comes with inherent problems, such as provision of poor quality, inaccurate, irrelevant or fraudulent information. As part of the OpenData.cz initiative, we are developing projects which will enable creation, maintenance, and usage of the data infrastructure formed by the Czech governmental linked open data. In particular, the project ODCleanStore will enable data consumers seamless automated data aggregation to simplify the manual aggregation process, which would have to be performed otherwise, and will also provide provenance tracking and justifications why the aggregated data should be trusted by the consumer in the given situation. In this paper, we describe two crucial aspects of the data aggregation process in ODCleanStore—resolution of data conflicts and computation of aggregate quality helping consumers to decide whether the aggregated data are worth using. Since the data aggregation algorithm is executed during query time, we show that the proposed algorithm is fast enough to work in real-world settings.

Keywords: open data; linked data; data aggregation; data quality; conflict resolution; data trustworthiness;

I. INTRODUCTION

All over the world, governments are connecting to the uprising trend of publishing governmental data as open data. If open data is original non-aggregated machine readable data which is freely available to everyone, anytime, and for whatever purpose. As a result, citizens paying the government will be able to see and analyze the performance of the government by observing the raw data or using third-party applications visualizing the data; companies will be able to use the data in arbitrary way to run their business.

Cannot we do more than just opening the data to simplify the data exploration and creation of applications on top of open data? If HTTP URLs were used as global identifiers for resources (e.g. things, persons, etc.), the data about these resources could be then exposed on these URLs and data consumers could use the current Web infrastructure to obtain relevant information about any resource by simply inserting the URL of the resource to the browser. Furthermore, if the open data were represented as RDF triples² we could link data (e.g. a public contract at http://isvzus.cz) to other data (e.g. a supplier of the contract held in Business Register) and, thus, create a huge web of interconnected data. The idea described is precisely the idea of linked data [3].

The linked data approach accelerates the evolution of the Web into an exponentially growing information space (the linked open data (LOD) cloud⁴), where the unprecedented volume of resources will offer a level of information integration and aggregation that has up to now not been possible. Indiscriminate addition of information, however, comes with inherent problems such as the provision of poor quality, inaccurate, irrelevant or fraudulent information. All will come with an associate cost of the data aggregation which will ultimately affect data consumer’s benefit and linked data applications usage and uptake. To enable seamless data aggregation and to simplify the data consumer’s judgement whether to trust the consumed data or not, we are developing as part of the OpenData.cz initiative⁴ several projects (modules) enabling creation, maintenance, and usage of the linked open data infrastructure (see Figure 1).

Since there is not much linked open data in the Czech Republic so far, the data acquisition module (see Figure 1) obtains Czech governmental data from various sources, such as (X)HTML pages or Excel spreadsheets, and convert it to RDF data. Currently, we are collecting data about public contracts, for which we also created an ontology⁵ http://code.google.com/p/public-contracts-ontology/ referred in the following text with the prefix pc:

http://opendatahandbook.org/en/

http://opendatahandbook.org/en/

http://code.google.com/p/public-contracts-ontology/

http://isvzus.cz
to help consumers decide which data is worth using. *Data visualization and analysis* module simplifies work of application developers and citizens by enabling to define which linked data returned by ODCS should be visualized, how, and whether an analysis of the data should be conducted (e.g. computing average salary in a company).

In this paper, we focus on the data aggregation process in ODCS, in particular on (1) the resolution of conflicts among the aggregated data and (2) how the aggregate quality of the data is computed. These two points present the main contributions of the paper.

In Section III we introduce the ODCS’s interfaces and describe briefly how data is cleaned, linked, scored, and stored. Section III describes the data aggregation algorithm – resolution of conflicts and computation of the aggregate quality. Section IV contains experiments; Section V related work and Section VI conclusions.

II. ODCCleanStore Project

RDF data are represented as typed statements – *triples* – consisting of a subject, a predicate and an object. The RDF data model can be viewed as a directed graph where edges, labeled with a predicate, lead from a subject to an object. Let \( U \) denote all possible elements (vertices, edges) in such a graph – *URI resources* (subjects, predicates, or objects), string values, optionally typed, called *literals* (only objects) and *blank nodes* (subjects or objects). Thus, a subject \( s \), a predicate \( p \) and an object \( o \) together form a *triple* \((s, p, o) \in U^3\). A triple may be part of a *named graph* – a set of triples identified by an URI; triples can be then extended to *quads* \((s, p, o, g) \in Q\) where \( g \in G \) is the named graph (its URI) to which the data belongs; when talking about “data in the named graph \( g \)”, we mean all the quads \((s, p, o, g)\).

In ODCS RDF data is stored in the form of quads.

The interfaces of ODCS are depicted in Figure 1. Data is submitted to ODCS by using a *web service for publishers*; it can be send directly as RDF data or obtained from the data acquisition module. The RDF data is stored to OpenLink Virtuoso.\(^6\) We use two important databases to store the data – dirty database (for incoming data) and clean database (for cleaned, linked and scored data available for consumers).

RDF data can be consumed in several ways (see Figure 1) using a *web service for consumers*; it can be consumed by the data visualization and analysis module or by third-party applications. In this paper, we discuss only querying by URI\(^7\) i.e. a consumer sends URI of a concept as part of her query (Query 1 in Figure 1) and as a response she gets the description of the desired concept aggregated from all available sources together with the aggregate quality; the sources of the data are accompanied with provenance metadata. Optionally, the consumer can specify conflict resolution policies customizing the aggregation process described in Section III. Moreover, the consumer can obtain (see Query 2 in Figure 1) an explanation how the quality of the particular quad returned by Query 1 was computed, which helps her to justify why she should trust the quad\(^8\).

A. Cleaning Data

When a new named graph \( g \) arrives to the dirty database, it is consequently cleaned, linked, scored and stored to the clean database. Given provenance metadata about \( g \) are stored separately (the score of \( g \) doesn’t apply to it).

1) Cleaners: The goal of custom cleaners is to correct errors in the incoming named graph \( g \) and normalize data for more accurate aggregation. Cleaners are custom Java classes implementing a defined interface. Both input and output of a cleaner is a named graph. For example, a custom cleaner may correct typos in the name of a month, or convert date to a given standard format.

2) Linkers: Since data describing the same concepts (e.g. persons, public contracts) can be identified by various identifiers (URIs), the application will support specification of rules, which will apply to the named graph \( g \) and try to reveal whether the data in \( g \) involve a new concept or a concept already present in the clean database; in the latter case, the application will create a link (\( \text{owl:sameAs} \) predicate) specifying that the given two URIs are representing the same concept. Linkers will also support creation of other types of links among data and links to other datasets (e.g. the LOD cloud). We use Silk engine\(^9\) and its specification language\(^10\) to specify policies driving the creation of links.

3) Quality Assessment: The ability to assess the *information quality* (IQ) of the consumed data presents one of the most important aspects of the information aggregation on the Web and will play a fundamental role in the adoption of Linked Data principles\(^9\). IQ is usually described by a series of *IQ dimensions* representing a set of desirable characteristics for an information resource\(^1\)\(^9\).

Currently, the Quality Assessment (QA) component just checks whether \( g \) satisfies custom consistency policies and based on that computes quality score of the graph. Sample consistency policies are: “The date held by the predicate \( \text{pc:awardDate} \) is later than the date held by the predicate \( \text{pc:publicationDate} \)”, “The predicate \( \text{pc:referenceNumber} \) has a value satisfying the given regular expression”, “Property \( \text{pc:actualPriceNumber} \) exists”. Policies may also be automatically generated from ontologies.

Let us introduce a function \( s_{ng} : G \rightarrow [0, 1]\); \( s_{ng}(g) \) computes the score of the incoming named graph \( g \), which

---

\(^{6}\)http://virtuoso.openlinksw.com/

\(^{7}\)http://purl.org/procurement/public-contracts#

\(^{8}\)Other types of queries will be also implemented in the future, such as keywords search or SPARQL query

\(^{10}\)Prefix pc: refers to http://purl.org/procurement/public-contracts#
is based on the application of the consistency policies on the graph \( g \) (in future, it will also take into account outputs of cleaners and linkers). Every policy \( r \) is associated with its weight \( w(r) \in [-1, 1] \); if the weight is positive (negative), the application of the policy to the graph \( g \) increases (decreases) the default score \( \beta \in (0, 1] \), i.e.:

\[
s_{ng}(g) = \min(\beta \cdot \prod_{r \in R} (1 + w(r)), 1)
\]

Furthermore, \( p = pub(g) \) denotes the publisher \( p \in P \) of the named graph \( g \) (e.g. \url{http://isvzu.cz}); \( P = \{ pub(g) \mid g \in G \} \) is the set of publishers. A score of the publisher \( p \) is computed with function \( s_p : P \to [0, 1] \) as the weighted average of scores \( s_{ng} \) of named graphs published by \( p \) weighted by \( |g| \), the number of triples in \( g \):

\[
s_p(p) = \frac{\sum_{g \in G | pub(g) = p} |g| \cdot s_{ng}(g)}{\sum_{g \in G | pub(g) = p} |g|}
\]

Based on that, the quality score of the graph \( g \) is computed by the function \( s : G \to [0, 1] \) as a weighted average of scores \( s_{ng} \) and \( s_p \), where \( \gamma \in [0, 1] \) is a constant:

\[
s(g) = \gamma \cdot s_{ng}(g) + (1 - \gamma) \cdot s_p(pub(g))
\]

Finally, all quads \((*,*,*,g)\) and the quality score \( s(g) \) are stored to the clean database. In the future, the QA component will assess a vector of quality scores, each score corresponding to evaluation of one IQ dimension.

III. DATA AGGREGATION COMPONENT

When aggregating data from various sources (source named graphs), e.g. \( g_1, g_2 \in G \), ODCS has to deal with instance level conflicts (conflicting values), which happen when quads \((s, p, o_1, g)\) and \((s, p, o_2, g)\) have inconsistent values \( o_1, o_2 \) for a certain predicate \( p \) and subject \( s \) \([12]\). Instance level conflicts can be solved at design time (while filling up the clean database) or query time. Since one of the basic tenets of the Web is the AAA slogan – Anyone can say Anything about Any topic – and we support the data consumer’s rights to select which data is worth using in the situation at her hand, we cannot deliberately aggregate the data as they are coming to the clean database. Therefore, the aggregation algorithm run during query time is as follows:

1) Replace URIs of resources representing the same concept (i.e. connected by an \( \text{owl:sameAs} \) path) with a single URI.
2) Remove duplicate quads.
3) Group quads to sets of conflicting quads.
4) For each set of conflicting quads:
   4.1) Choose and apply an aggregation method; yields to one or more aggregated values.
   4.2) Compute aggregate quality for aggregated values.
   4.3) Create resulting aggregated quads enriched with the aggregate quality and sources of each quad.

The input to the data aggregation process is (1) a collection of quads from the clean database to be aggregated, (2) \( \text{owl:sameAs} \) links between URI resources occurring in the quads, (3) aggregation settings (e.g. aggregation methods for predicates), and (4) quality scores for named graphs of the quads. The output is a collection of aggregated quads enriched with the aggregate quality and sources for each quad; the sources are accompanied with provenance data.

When aggregating data, statements can be expressed using different predicates, although having similar or the same semantics. ODCS maintains ontologies describing the stored data and enables the creation of mappings between them (e.g. between predicates with the same meaning as \( \text{owl:sameAs} \) links), using both manual creation and Silk; these mappings are taken into account before entering Step 1 of the data aggregation algorithm – predicates are mapped to the preferred predicates (those in the consumer’s query).

A. Implicit Conflict Resolution

Steps 1-3 of the aggregation algorithm – implicit conflict resolution – do not depend on the chosen aggregation settings; it prepares input data for Step 4 so that conflicts and quality can be computed independently on chosen resource URIs or overlapping older versions of data. We describe it...
Algorithm 1 Implicit conflict resolution algorithm

1: Create graph \( H = (V, E) \) from the given \texttt{owl:sameAs} links; edges \( E \) are the \texttt{owl:sameAs} predicates, vertices \( V \subset U \) the URI resources they are connecting.
2: Find the set of weakly connected components \( C \) in \( H \).
3: For each connected component \( C \in \mathcal{C} \) do
4: Choose \( uri(C) \), a single URI from the component, preferring URIs given in the consumer’s query.
5: end for
6: For each input quad \((s, p, o, g)\) do
7: Replace \( s \) with \( uri(C_s) \), \( s \in C_s \), \( C_s \in \mathcal{C} \). Do the same with predicate \( p \) and object \( o \).
8: end for
9: Remove duplicate quads that might have appeared.
10: For each quad \( q = (s, p, o, g_1) \) do
11: If there is another quad \((s, p, o, g_2)\) where named graph \( g_2 \) is an update of \( g_1 \), remove \( q \).
12: end for
13: Group quads into sets of conflicting quads \( Q_{s,p} \), i.e. having the same subject \( s \) and predicate \( p \).

in detail in Algorithm[1] The complexity of Algorithm[1] is \( O(S \log S + N \log N) \), \( S \) is the number of \texttt{owl:sameAs} links, \( N \) the number of input quads.

Since data acquisition is realized mainly by automated (X)HTML page extractors, data obtained from one source may change over time. It is important to keep track of these changes so that the consumer is given the latest data and provenance metadata but at the same time older data, which might have disappeared from the web, are retained. ODCS detects whether a named graph is an update of another graph, however, the detection is beyond the scope of this paper.

B. Application of an Aggregation Method (Step 4.1)

Step 4 is applied to each set of conflicting quads \( Q_{s,p} = \{q_1, \ldots, q_n\} \), \( q_i = (s, p, o_i, g_i) \), \( i \in \{1, \ldots, n\} \); thus, the context of conflict resolution is given by the collection of objects \( o_i \); subject \( s \) and predicate \( p \) are constant for one execution of Step 4. Let us denote \( V = (v_1, \ldots, v_n) \) the collection of object values \( v_i = o_i \) from the named graph \( g_i \); further in the text, \( v_i \) and \( g_i \) are always as defined here.

In Step 4.1, a set of conflicting quads \( Q_{s,p} \) is aggregated by the application of an aggregation method defined for the predicate \( p \) in the aggregation settings; each such application selects one or more values \( v_i \) or computes new aggregated values; let us introduce the set \( A \) holding such selected or computed values. In order to provide maximum flexibility, aggregation methods can be customized for each predicate by the conflict resolution policies submitted by a consumer in Query 1 in Figure[1] If no aggregation method is given for a predicate, a default method is applied. The available aggregation methods are based on aggregations commonly used in database conflict resolution strategies[2]:

- ANY,MIN,MAX,SHORTEST, LONGEST – an arbitrary value, minimum, maximum, shortest, or longest is selected from the conflicting values \( V \)
- AVG, MEDIAN, CONCAT – computes the average, median, or concatenation of conflicting values
- BEST – the value with the highest aggregate quality (see Section[III-C]) is selected
- LATEST – the value with the newest time is selected
- ALL – all input values are preserved

If the aggregation method cannot be applied to a value (e.g. an average of a string literal), the behavior depends on an aggregation error strategy given in the aggregation settings – the value may be either discarded, or included in the output without aggregation. In the current implementation, only conflicts in the place of the object are resolved. Resolution of conflicts could be extended to subjects analogously.

C. Computation of the Aggregate Quality (Step 4.2)

In Step 4.2, we compute the aggregate quality of values \( v \in A \), denoted \( q(v) \). The set of graphs that agree on a value \( v \in V \) is denoted \( agree(v) = \{g_i \mid v_i = v\} \). Multiple real-world cases (such as the one in Section[IV]) lead us to three factors of the aggregate quality computation: quality scores of the source named graphs \( g_i \), the size of \( agree(v) \), and the difference between value \( v \) and other (conflicting) values from \( V \). The three factors are further accompanied by constraints that should be satisfied by function \( q \). The most important constraints can be outlined as:

1) If there is a named graph \( g \) asserting a non-conflicting value \( v \), the aggregate quality (based just on the value \( v \) should be at least \( s(g) \).
2) \( q(v) \) is increasing with quality scores of source named graphs \( v \) was selected from or calculated from.
3) \( q(v) \) is decreasing with difference of other values \( v_i \in V \), taking their quality scores \( s(g_i) \) into consideration.
4) If multiple sources agree on the same value, the aggregate quality is increased.

1) First Quality Factor – Scores of Sources: First, we calculate aggregate quality \( q_1(v) \) based on the quality scores of the sources. A value \( v \in V \) may be calculated from all the sources (aggregations AVG, MEDIAN, CONCAT), or come from named graphs containing a quad \((s, p, v, g_i)\) (other aggregations). In the former case we use formula (a), in the latter case we use formula (b) (in accordance with Constraint[1]):

\[
q_1(v) = \begin{cases} 
\text{avg} \{s(g) \mid g \in \{g_1, \ldots, g_n\}\} & \text{(a)} \\
\max \{s(g) \mid g \in agree(v)\} & \text{(b)}
\end{cases}
\]

2) Second Quality Factor – Conflicting Values: In the second step, we compute aggregate quality \( q_2(v) \) based on \( q_1(v) \) and differences of conflicting values \( V \). For this, we use a metric \( d : U \times U \to [0,1] \) satisfying \( d(v, v) = 0 \).
We use $d(x, y) = |(x - y) / \text{avg}(x, y)|$ in case of numeric literals, normalized Levenshtein distance in case of string literals and $d(x, y) = 1$, where $x \neq y$, for URI resources and nodes of a different type. Other metrics can be added for different types of literal values (such as $\text{xsd:}\text{date}$).

If there are conflicting values different from $v$, the aggregate quality of $v$ is reduced increasingly with the value of metric $d$ and the score of the source of the conflicting value (Constraint 2):

$$q_2(v) = q_1(v) \cdot (1 - \frac{\sum_{i=1}^{n} s(g_i) d(v, v_i)}{\sum_{i=1}^{n} s(g_i)})$$

Decreasing aggregate quality due to conflicting values is not always the right solution, however. For example, predicate rdf:type often has multiple valid values which are not in fact conflicting. In order to handle this case, the consumer can set a parameter called multivalue in the aggregation settings which instructs the aggregation method to use $q_2(v) \equiv q_1(v)$ instead. Multivalue can be set both globally and individually for each predicate.

3) Third Quality Factor: Confirmation by Multiple Sources: Intuitively, if multiple different sources agree on a single value, we should trust this value more than each of the sources individually. We reflect this in the final phase of aggregate quality computation $q_3(v)$ ($C \in \mathbb{N}$ is a constant):

$$q_3(v) = q_2(v) +$$

$$+ (1 - q_2(v)) \cdot \min\left(\frac{-q_1(v) + \sum_{g \in \text{aggregated}} s(g)}{C}, 1\right)$$

In case of the aggregation method CONCAT, the aggregate quality is computed as $q(v) \equiv q_1(v)$; in case of AVG and MEDIAN, $q(v) \equiv q_2(v)$ if $\forall i \in \{1, \ldots, n\}: v \neq v_i$; otherwise and for all other aggregation methods $q(v) = q_3(v)$, satisfying Constraints 1 – 4.

D. Preparing Resulting Aggregated Data (Step 4.3)

Step 4.3 creates resulting quads enriched with aggregate quality, source named graphs, and provenance metadata for these graphs. Thus, the resulting RDF data of Step 4 (and of Query 1 in Figure 1) includes quads $(s, p, v, g_v)$ (where $v \in A$ are aggregated values, $g_v$ new unique named graphs), triples $(g_v, x:aQuality, q(v))$ holding the aggregate quality and $(g_v, x:sourceGraph, g_i)$ holding the source named graphs of value $v$ and provenance metadata for each $g_i$ in the response.

The complexity of the aggregate quality computation for a fixed $v \in A$ is $O(|V| \cdot D)$; $D$ is the complexity of the distance metric evaluation. This gives us the overall complexity of Step 4 for one $Q_{s,p}: O(|V|^2 \cdot D)$ for ALL and BEST, $O(|V| \log |V| + |V|^2 \cdot D)$ for MEDIAN; $O(|V|^2 \cdot D)$ for others.

### IV. Experiments

Let us start with an illustrative example; suppose in the clean database, there are data about Berlin coming from multiple sources, e.g. DBpedia, GeoNames, Freebase, etc and we request the aggregated information about Berlin.

Firstly, we have to solve the use of different URIs identifying Berlin. This problem is handled in ODCS by linkers that would be able to add owl:sameAs links between these URIs e.g. based on the similarity of their labels and geographical location. Another problem is that different predicates are used to express geographical location; to solve this, the respective ontologies will be loaded into ODCS and mappings between ontologies would be created.

Having ODCS translate URIs for us, we can aggregate results. A different aggregation method may be suitable for each predicate – we may choose AVG for geo:long, BEST for rdfs:label, ALL for dbprop:population. We intentionally added an error stating that geo:lat of Berlin is 13 (instead of 52). We suppose the score $s$ is set to 0.9 for DBpedia and to 0.8 for other sources, the multivalue parameter of the aggregation settings is set for rdfs:label and rdf:type.

Table I gives the results of the data aggregation algorithm. The aggregate quality of dbprop:population is decreased because there were different conflicting values; the quality of label “Berlin” is very high, as all sources agree on it; the quality of geo:lat is significantly lower, because of the introduced erroneous value.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Value</th>
<th>Source</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:label</td>
<td>Berlin</td>
<td>all</td>
<td>0.963</td>
</tr>
<tr>
<td>rdf:type</td>
<td>dbp:City</td>
<td>DBpedia</td>
<td>0.900</td>
</tr>
<tr>
<td>dbprop:population</td>
<td>3450889</td>
<td>DBpedia</td>
<td>0.897</td>
</tr>
<tr>
<td>dbprop:population</td>
<td>3426354</td>
<td>GeoNames</td>
<td>0.797</td>
</tr>
<tr>
<td>geo:long</td>
<td>13.4074</td>
<td>all</td>
<td>0.833</td>
</tr>
<tr>
<td>geo:lat</td>
<td>42.7402</td>
<td>all</td>
<td>0.497</td>
</tr>
</tbody>
</table>

12Predicate rdf:type states that a resource belongs to a class.
13Predicates x:aQuality and x:sourceGraph are defined by the ontology internally used by ODCS.
14http://goo.gl/n8PKk
15http://dbpedia.org/resource/Berlin
16http://sws.geonames.org/2950159/
17http://rdf.freebase.com/ns/en.bangalore
18http://dbpedia.org/downloads32#h72-1
19Hardware configuration: Intel Core2 Duo 2x2.4 Hz, 3 GB RAM
Table II

<table>
<thead>
<tr>
<th>Triples</th>
<th>Aggregation</th>
<th>Multiverse</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100,000</td>
<td>ALL</td>
<td>no</td>
<td>1.75 s</td>
</tr>
<tr>
<td>100,000</td>
<td>ANY</td>
<td>no</td>
<td>1.02 s</td>
</tr>
<tr>
<td>100,000</td>
<td>ALL</td>
<td>yes</td>
<td>1.01 s</td>
</tr>
<tr>
<td>100,000</td>
<td>CONCAT</td>
<td>yes</td>
<td>0.96 s</td>
</tr>
<tr>
<td>100,000</td>
<td>ANY</td>
<td>yes</td>
<td>0.83 s</td>
</tr>
</tbody>
</table>

The number of conflicting values will be typically small, small 100s of sets of conflicting values with size in 10s of values for a query. Therefore, our experiment gives us very reasonable times per a conflicting set and has demonstrated that it is indeed fast enough to work in real-world settings.

V. RELATED WORK

Lots of works deal with instance level conflicts during the design time, e.g. [5] [9], however, this approach is not suitable, because the worthiness of the data depends on the situation at the consumer’s hand.

To the best of our knowledge, there is just one another fusion software aggregating RDF data which is currently under development [8]. Sieve is part of Linked Data Integration Framework (LDIF) [10]; however, the purpose of LDIF is different – it uses Sieve to aggregate data while being stored to the clean database (not during execution of queries as in ODCS), which may be suitable for closed corporate environments, where the desired data are known in advance, but is not sufficient for open Web environments, where every consumer has different requirements on the aggregated data (or is looking for raw data “aggregated” by method ALL). Furthermore, Sieve does not compute aggregate quality, taking into account the constraints presented; the aggregation of provenance information is also not well described. On the other hand, Sieve allows to customize aggregation method used, which we plan in the near future.

Aurora [11] is an integration system of heterogenous non-RDF data; its query model [12] enriches the SQL SELECT by enabling to define attribute and record conflict resolution functions. ODCS offers more built-in aggregation methods, on the other hand, Aurora allows user defined attribute aggregation functions; in ODCS, record conflicts are either discovered by linkers, or there is no record conflict.

VI. CONCLUSIONS

In this paper, we described the needs for data aggregation tools which will help data consumers to aggregate large volumes of linked open data. We particularized vital aspects of the data aggregation algorithm – (1) solving conflicts among data and (2) computing aggregate quality. We demonstrated that the aggregation algorithm is fast enough to work in real-world settings – 2500 sets of conflicting triples, i.e. more than a typical request “Give me all information about the concept X” triggers, has response time 1.75s in the most complicated case.

The aggregate quality, as described in this paper, is objective, situation independent (although the actual value returned from the conflict resolution process may be influenced by the conflict resolution policies). In the future, we will extend conflict resolution by taking into account consumers’ subjective provenance policies, such as “Trust data hosted by [http://isvzus.cz]”.

VII. ACKNOWLEDGMENTS

The work presented in this article has been funded in part by the Czech Science Foundation (GACR), grant number P202/10/0573, GAUK 3110, SVV-2012-265312, and EU ICT FP7 under No.257943 (LOD2 project).

REFERENCES