Triplify – Light-Weight Linked Data Publication from Relational Databases

Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, David Aumueller
Universität Leipzig, Postfach 100920, 04009 Leipzig, Germany
{auer|dietzold|lehmann|hellmann|aumueller}@informatik.uni-leipzig.de

ABSTRACT
In this paper we present Triplify – a simplistic but effective approach to publish Linked Data from relational databases. Triplify is based on mapping HTTP-URI requests onto relational database queries. Triplify transforms the resulting relations into RDF statements and publishes the data on the Web in various RDF serializations, in particular as Linked Data. The rationale for developing Triplify is that the largest part of information on the Web is already stored in structured form, often as data contained in relational databases, but usually published by Web applications only as HTML mixing structure, layout, and content. In order to reveal the pure structured information behind the current Web, we have implemented Triplify as a light-weight software component, which can be easily integrated into and deployed by the numerous, widely installed Web applications. Our approach includes a method for publishing update logs to enable incremental crawling of linked data sources. Triplify is complemented by a library of configurations for common relational schemata and a REST-enabled data source registry. Triplify configurations containing mappings are provided for many popular Web applications, including osCommerce, WordPress, Drupal, Gallery, and phpBB. We will show that despite its light-weight architecture Triplify is usable to publish very large datasets, such as 160GB of geo data from the OpenStreetMap project.

Categories and Subject Descriptors
H.3.4 [Systems and Software]: Distributed systems; H.3.5 [Online Information Services]: Data sharing

General Terms
Algorithms, Languages

Keywords
Data Web, Databases, Geo data, Linked Data, RDF, SQL, Semantic Web, Web application

1. INTRODUCTION
Even though significant research and development efforts have been undertaken, the vision of an ubiquitous Semantic Web has not yet become reality. The growth of semantic representations is (as we will show in the next section) still outpaced by the growth of traditional Web pages and one might remain skeptical about the potential success of the Semantic Web in general. The missing spark for expanding the Semantic Web is to overcome the chicken-and-egg dilemma between missing semantic representations and search facilities on the Web.

In this paper we, therefore, present Triplify – an approach to leverage relational representations behind existing Web applications. The vast majority of Web content is already generated by database-driven Web applications. These applications are often implemented in scripting languages such as PHP, Perl, Python, or Ruby. Almost always relational databases are used to store data persistently. The most popular DBMS for Web applications is MySQL. However, the structure and semantics encoded in relational database schemes are unfortunately often neither accessible to Web search engines and mashups nor available for Web data integration.

Aiming at a larger deployment of semantic technologies on the Web with Triplify, we specifically intend to:

- enable Web developers to publish easily RDF triples, Linked Data, JSON, or CSV from existing Web applications,
- offer pre-configured mappings to a variety of popular Web applications such as WordPress, Gallery, and Drupal,
- allow data harvesters to retrieve selectively updates to published content without the need to re-crawl unchanged content,
- showcase the flexibility and scalability of the approach with the example of 160 GB of semantically annotated geo data that is published from the OpenStreetMap project.

The importance of revealing relational data and making it available as RDF and, more recently, as Linked Data [2, 3] has been recognized already. Most notably, Virtuoso RDF views [5, 9] and D2RQ [4] are production-ready tools for generating RDF representations from relational database content. A variety of other approaches has been presented recently (cf. Section 6). Some of them even aim at automating partially the generation of suitable mappings from relations to RDF vocabularies. However, the growth of semantic representations on the Web still lacks sufficient momentum. From our point of view, a major reason for the lack of deployment of these tools and approaches lies in the complexity
One of the ideas behind Triplify is to enable a large number of applications and people to take part in the Semantic Web. We argue that the number of different stakeholders needs to increase in order to realise the vision of the Semantic Web. This means in particular that we need not only large knowledge bases, but also data created by many small data publishers. Although it is difficult to give a rigorous proof for such a claim, we observed a few interesting facts, when analysing the growth of the Semantic Web, in particular in relation to the World Wide Web.

Some statistics are summarized in Figure 2. The red line in the figure depicts the growth of the WWW measured by the number of indexed pages on major search engines. This estimate is computed by extrapolating the total number of pages in a search engine index from known or computed word frequencies of common words. Along the line of similar studies, the statistics suggest an exponential growth of pages on the WWW. The number of hosts (green line) grows slower, while still being roughly exponential.

We tried to relate this to the growth of the Semantic Web. The currently most complete index of Semantic Web data is probably Sindice. It stores 37.72 million documents, which accounts for slightly more than 0.1% of all WWW documents. Since the growth of documents in Sindice was closely related to upgrades in their technical infrastructure in the past, we cannot reliably use their growth rate. However, the absolute number indicates that semantic representations are not yet common in today’s Web.

Another semantic search engine is Swoogle. The blue

1see http://www.worldwidewebsize.com
2statistics from http://isc.org
3see http://www.isc.org/ops/ds/hosts.png for a long term analysis
4http://sindice.com
5http://swoogle.umbc.edu/
Figure 1: Temporal comparison of different growth indicators for the traditional WWW and the Semantic Data Web. The first and second lines show statistics for indexed web pages (obtained from WorldWideWebSize.com) and hosts (obtained from isc.org) for the WWW. The third and fourth lines are counts of Semantic Web documents from Swoogle (obtained from Swoogle publications and swoogle.umbc.edu) and Ping The Semantic Web (obtained by log file analysis). Please note the different order of magnitude for each line.

line in Figure 2 illustrates the growth in Semantic Web documents (SWDs) in the Swoogle index. It indexed 1.5 million documents from the start of Swoogle early in 2005 until the middle of 2006. Currently (October 2008), 2.7 million documents are indexed. Note that Swoogle, unlike Sindice, does not crawl large ontologies as present in the Linking Open Data (LOD) cloud. If we take this aspect into account, the Semantic Web growth curve would be steeper. However, the LOD data sets are usually very large data sets extracted from single sources. A growth of the LOD cloud does not indicate that the number of different participants in the Semantic Web would itself be high.

We also analysed Ping the Semantic Web (PTSW)\textsuperscript{6} log files since they started logging (last line). Apart from the startup phase (i.e. all documents collected so far fall in the first months), we observed an approximately linear growth of about 15,000 new resources pinged each month over the following 14 months. Overall, our observations – while they have to be interpreted very carefully – indicate that the number of data publishers in the Semantic Web is still some orders of magnitudes lower than those in the WWW. Furthermore, its relatively low (estimated) growth rate shows that simple ways need to be found for a larger number of web applications to take part in the Semantic Web.

A recent study \cite{7} analysed how terms (classes and properties) depend on each other in the current Semantic Web. They used a large data set collected by the Falcons semantic search engine\textsuperscript{7} consisting of more than 1.2 million terms and 3,000 vocabularies. By analysing the term dependency graph, they concluded that the “schema-level of the Semantic Web is still far away from a Web of interlinked ontologies, which indicates that ontologies are rarely reused and it will lead to difficulties for data integration.”. Along similar lines, \cite{8} analysed a corpus of 1.7 million Semantic Web documents with more than 300 million triples. They found that 95% of the terms are not used on an instance level. Most terms are from meta-level (RDF, RDFS, OWL) or a few popular (FOAF, DC, RSS) ontologies. By enabling users to map on existing vocabularies, Triplify encourages the re-use of terms and will thereby contribute positively to data integration on the Semantic Web even on the schema level.

3. THE CONCEPT

Triplify is based on the definition of relational database views for a specific Web application (or more general for a specific relational database schema) in order to retrieve valuable information and to convert the results of these queries into RDF, JSON and Linked Data. Our experiences showed that for most Web applications a relatively small number of simple queries (usually between 5 to 20) is sufficient to extract the important public information contained in a database. After generating such views, the Triplify tool is used to convert them into an RDF, JSON or Linked Data representation (explained in more detail later), which can be shared and accessed on the Web. The generation of these semantic representations can be performed on demand or in advance (ETL). An overview of Triplify in the context of current Web technologies is depicted in Figure 2.

3.1 Triplify Configuration

At the core of the Triplify concept is the definition of a configuration adapted for a certain database schema. A configuration contains all the information the Triplify implementation needs in order to generate sets of RDF triples which are returned on a URI requests it receives via HTTP. The Triplify configuration is defined as follows:

**Definition 1 (Triplify configuration)** A Triplify configuration is a triple \((s, \phi, \mu)\) where:

- \(s\) is a default schema namespace,
- \(\phi\) is a mapping of namespace prefixes to namespace URIs,
The use of SQL as a mapping language has many advantages compared with employing newly developed mapping languages:

- **SQL** is a mature language, which was specifically developed to transform between relational structures (results of queries are also relational structures). Hence, SQL supports many features, which are currently not available in other DB to RDF mapping approaches. Examples are aggregation and grouping functions or complex joins.

- Being based on SQL views allows Triplify to push almost all expensive operations down to the database. Databases are heavily optimized for data indexing and retrieval, thus positively affecting the overall scalability of Triplify.

- Nearly all software developers and many server administrators are proficient in SQL. Using Triplify barely requires any additional knowledge and allows users to employ semantic technologies, while working in a familiar environment.

### 3.2 Relational View Structure

For converting SQL query results into the RDF data model, Triplify follows a *multiple-table-to-class* approach preceded by a relational data transformation. Hence, the results of the relational data transformation require a certain structure in order to be representable in RDF:

- The first column of each view must contain an identifier which will be used to generate instance URIs. This identifier can be, for example, the primary key of the database table.

- Column names of the resulting views will be used to generate property URIs.

- Individual cells of the query result contain data values or references to other instances and will eventually constitute the objects of resulting triples.

In order to be able to convert the SQL views defined in the Triplify configuration into the RDF data model, the views can be annotated inline using some extensions to SQL, which remain transparent to the SQL processor, but influence the query result, in particular the column names of the returned relations.

When renaming query result column names, the following extensions to SQL can be used to direct Triplify to employ existing RDF vocabularies and to export characteristics of the triple objects obtained from the respective column:

- **Mapping to existing vocabularies.** By renaming the columns of the query result, properties from existing vocabularies can be easily reused. To facilitate readability, the namespace prefixes defined in the Triplify configuration in φ can be used. In the following query, for example, values from the name column from a users table are mapped to the name property of the FOAF namespace:

  ```sql
  SELECT id, name AS 'foaf:name' FROM users
  ```

- **Object properties.** By default all properties are considered to be datatype properties and cells from the result set are converted into RDF literals. By appending a reference to another instance set separated with '>' to the column name, Triplify will use the column values to generate URIs (i.e. RDF links).

  ```sql
  SELECT id, name AS 'foaf:name' FROM users
  ```

- **Datatypes.** Triplify uses SQL introspection to retrieve automatically the datatype of a certain column and create RDF literals with matching XSD datatypes. In order to tweak this behaviour, the same technique of appending hints to column names can be used for instructing Triplify to generate RDF literals of a certain datatype. The used separator is '^^'.

  ```sql
  SELECT id, name AS 'foaf:name' FROM users
  ```

- **Language tags.** All string literals resulting from a result set column can be tagged with a language tag that is appended to the column name separated with '@'.

The use of these SQL extensions is illustrated in the following example:
Figure 3: Mapping µ from URL patterns to SQL query (sets) in the Triplify configuration for the WordPress blogging system (PHP code).

Example 1 (SQL extensions) The following query, illustrating the Triplify SQL extensions, selects information from a products table. The used Triplify column naming extensions will result in the creation of literals of type xsd:decimal for the price column, in the mapping of the values in the desc column to literals with 'en' language tag attached to rdfs:label properties and in object property instances of the property belongsToCategory referring to category instances for values in the cat column.

SELECT id, price AS 'price^^xsd:decimal', desc AS 'rdfs:label@en', cat AS 'belongsToCategory->category' FROM products

3.3 Triple Extraction

The conversion of database content into RDF triples can be performed on demand (i.e. when a URL in the Triplify namespace is accessed) or in advance, according to the ETL paradigm (Extract-Transform-Load). In the first case, the Triplify script searches a matching URL pattern for the requested instance. In the second case, Triplify derives queries for retrieving lists of instances with those class names (base SQL queries) in the configuration. From the SQL queries associated with those class names (base SQL queries) in the configuration, Triplify generates Linked Data with the possibility to publish data on different levels of a URL hierarchy. On the top level Triplify publishes only links to classes (endpoint request). An URI of an endpoint request usually looks as follows: http://myblog.de/triplify/. From the class descriptions RDF links point to individual instances (class request). An URI of a class request would then look like this: http://myblog.de/triplify/posts. Individual instances from the classes could be finally accessed by appending the id of the instance, e.g.: http://myblog.de/triplify/posts/13.

In order to simplify this process of generating Linked Data, Triplify allows to use the class names as URL patterns in the Triplify configuration. From the SQL queries associated with those class names (base SQL queries) in the configuration, Triplify derives queries for retrieving lists of instances and for retrieving individual instance descriptions. The base SQL view just selects all relevant information about all instances. In order to obtain a query returning a list of instances, the base view is projected onto the first column (i.e. the id column). In order to obtain a query retrieving all relevant information about one individual instance, an SQL-where-clause restricting the query to the id of the requested instance is appended. In most cases this simplifies the creation of Triplify configurations enormously.

3.4 Linked Data Update Logs

When data is published on the Web, for example as Linked Data, it is important to keep track of data (and hence RDF) updates so that crawlers know what has changed (after the last crawl) and should be-retrieved from that endpoint. A centralized registry (such as implemented by PingTheSemanticWeb service\(^8\)) does not seem to be feasible, when Linked Data becomes more popular. A centralized service,
Algorithm 1: Triple Extraction Algorithm

Input: request URL, Triplify configuration
Output: set $T$ of triples

1. foreach URL pattern from Triplify configuration do
   2. if request URL represents endpoint request then
      3. $T = T \cup \{\text{RDF link to class request for URL pattern}\}$
   else
      4. if request URL matches URL pattern then
         5. foreach SQL query template associated with URL pattern do
            6. Query = replacePlaceholder(URL pattern, SQL query template, request URL);
            7. if request URL represents class request then
               8. Query = projectToFirstColumn (Query);
            else
               9. if request URL represents instance request then
                  10. Query = addWhereClause (Query, instanceId);
               Result = execute(Query);
               $T = T \cup \text{convert(Result)}$;
            end
         end
      end
   end
   15. return $T$

which also constitutes a Single Point of Failure, will hardly be able to handle millions of Linked Data endpoints pinging such a registry each time a small change occurs.

The approach Triplify follows are *Linked Data Update Logs*. Each Linked Data endpoint provides information about updates performed in a certain timespan as a special Linked Data source. Updates occurring within a certain timespan are grouped into nested update collections. The coarsest update collections cover years, which in turn contain update collections covering months, which again contain daily update collections and this process of nesting collections is continued until we reach update collections covering seconds in time. Update collections covering seconds are the most fine-grained update collections and contain RDF links to all Linked Data resources updated within this period of time. For very frequently updated LOD endpoints (e.g. Wikipedia) this interval of one second will be small enough, so the related update information can still be easily retrieved. For rarely updated LOD endpoints (e.g. a personal Weblog) links should only point to non-empty update collections in order to prevent crawlers from performing unnecessary HTTP requests. Individual updates are identified by a sequential identifier. Arbitrary metadata can be attached to these updates, such as the time of the update or a certain person who performed the update.

Example 2 (Nested Linked Data Update Log) Let us assume myBlog.de is a popular WordPress blog. The Triplify endpoint is reachable via http://myBlog.de/lod/. New blog posts, comments, and updates of existing ones are published in the special namespace below http://myBlog.de/lod/updates. Retrieving http://myBlog.de/lod/update,
4. IMPLEMENTATION

The Triplify concept has currently been implemented only in PHP, but implementations in other popular Web application languages are planned. The Triplify implementation was performed with the aim of producing a small, lightweight plugin for existing Web applications. The core of the implementation contains barely more than 300 lines of code, which simplifies integration into Web applications to a great extent. The Triplify implementation needs direct access to the relational database by means of either a PDO object (the standard database abstraction layer in PHP) or the MySQL driver. The Web applications into which Triplify is integrated may, however, use any other database abstraction framework. Triplify works with all relational databases accessible from PHP, since the only database vendor specific SQL code is contained in the actual Triplify configuration’s SQL views. When available, Triplify also uses URL rewriting, in order to produce concise RDF URLs. When SQL functions are insufficient or inconvenient for modifying data, user-defined call-back functions (implemented in PHP) allow to process data before values are exported to RDF literals. In some cases database schemas contain tables which have a property-value or even a subject-property-value structure. Triplify is able to deal with these cases by dynamically deriving the property name from a preceding column value instead of the current column name. Triplify allows to add arbitrary metadata (such as provenance or licensing information) to all generated RDF output.

The deployment of Triplify in conjunction with an existing Web application can be performed in two simple steps:

1. copying the Triplify script into the Web application’s directory.
2. a configuration matching the Web application’s database schema can be either downloaded from the Triplify configuration repository or has to be created manually.

Performance. Triplify was developed primarily for small to medium Web applications (i.e. less than 100MB database content). However, since Triplify’s application logic is simple and since nearly all workload is pushed down to the database, Triplify can also be used with very large databases (as we detail in Section 5). Furthermore, Triplify supports caching of the processed results in order to increase performance.

Privacy. Web developers who integrate Triplify into their applications should be cautious not to reveal any sensitive information. For example, email addresses should be SHA1 hashed and password hashes as well as information which is not publicly accessible should be omitted in the Triplify SQL queries. As a rule of thumb, only such information should be made available through Triplify, which is also publicly readable on Web pages.

Triplify is licensed under the terms of the GNU Lesser General Public License; everybody is free to copy, modify or redistribute it, even together with commercial software.

4.1 Configuration Repository

Most web applications are deployed many times with identical database schemas. Table 1 summarizes the most popular Web applications available at sourceforge.net and their monthly download figures. This data suggests that a relatively small amount of Web applications is deployed many hundred-thousand times on the Web. In order to reuse Triplify configurations for different deployments of the same database schema, configurations for popular Web applications are collected on the Triplify Wiki at http://triplify.org. Currently, most of these configurations still have to be installed manually, but our ultimate aim is to make Triplify a direct part of popular Web application distributions.

4.2 Endpoint Registry

Triplify just tackles one side of the chicken-and-egg problem of the Semantic Data Web — the availability of a critical mass of machine-readable data on the Web. The other side is to showcase how this data can be used to improve the user experience on the Web (in particular search, browsing and information integration). To facilitate such applications which incorporate Triplify data sources, we developed a light-weight endpoint registry. A new Linked Data endpoint can be conveniently registered by making a single HTTP request and appending the URL of the endpoint as a request parameter. This registry itself is available as a Triplify Linked Data endpoint at http://triplify.org/triplify.

5. EVALUATION

Our evaluation aims at demonstrating that Triplify is suited for the two application scenarios: (a) the semantification of existing small and medium Web applications such as Blogs, Wikis, online shops, Web forums or picture galleries and (b) the publication of large relational databases on the Web.

5.1 Integration into Web Applications

In order to evaluate our approach, we integrated Triplify into a number of different Web applications. The Triplify Web site\(^9\) includes a repository of these and other third-party contributed Triplify configurations for popular Web applications. General experiences are that creating a Triplify configuration for an average Web application (containing approximately 10–20 relations) takes around 2–3 hours including the search for existing vocabularies. We also noticed that existing vocabularies already cover the majority of the required cases. Of particular importance are the Friend-of-a-Friend (FOAF)\(^10\), Semantically Interlinked Online Communities (SIOC)\(^11\) and Dublin Core (DC)\(^12\) vocabularies.

In the remainder of this subsection we will report our experiences in creating Triplify configurations for the three Web applications osCommerce (online shop), phpBB (Web forum) and Gallery (Web photo album).

osCommerce. osCommerce Online Merchant is one of the most popular open-source online shop e-commerce solutions. The database schema of osCommerce contains almost 50 tables of which just 12 contain valuable public information. The osCommerce Triplify configuration consists of eight query sets (which are mapped to RDFS classes) and 13 queries. The following data structures are, in particular, exposed by Triplify: (1) a hierarchy of product categories, which uses vocabulary elements from the Simple Knowledge Organization System (SKOS) and Dublin Core, (2) a list of products and manufacturers based on the GoodRelations on-

\(^9\)http://Triplify.org

\(^10\)http://www.foaf-project.org

\(^11\)http://www.sioc-project.org

\(^12\)http://www.dublincore.org
... but complemented with additional attributes that are defined in a separate Triplify oCommerce namespace, and (3) a list of reviews for products represented as SIOC posts.

**phpBB.** According to download figures at Sourceforge.net, phpBB is the most popular open source Web forum solution. While the whole database schema consists of over 60 tables, the most relevant information about users, groups, and posts is located in just a few tables. The phpBB Triplify configuration consists of six query sets with one to two queries each. Given their good coverage, vocabulary elements from the SIOC and FOAF namespace are sufficient. The resulting RDF graph contains information about forum users (sioc:User) and groups (sioc:Usergroup) and the community content produced by the users (sioc:Forum and sioc:Post). In order to preserve the privacy of phpBB users, the Triplify configuration uses MySQL’s SHA function to encrypt users’ email addresses for storage in the sioc:email_sha1 attribute (cf. also the first query of the ‘user’ block in Figure 3).

**Gallery.** Gallery is a Web application allowing to publish and organize photos in albums. Metadata such as title and location can be attached to photos as well as community-supplied comments and ratings. Gallery’s database schema consists of over 40 tables out of which just six are used by Triplify. The Triplify configuration of Gallery consists of five query sets with one or two queries each. The resulting RDF graph uses vocabulary elements from the SIOC (both core and types module), FOAF and Dublin Core namespaces as well as some custom-defined ones. The latter include specific attributes for images and the number of views of an image in a gallery.

### 5.2 Publishing OpenStreetMap Geo Data

In addition to using Triplify for publishing RDF from the long tail of million of Web applications deployed, we evaluated the software with the very large datasets produced by the OpenStreetMap project. The OpenStreetMap project has successfully applied the Wiki approach to geo data. Thousands of users worldwide upload GPS traces from their navigation systems, mobile phones or GPS trackers to the OpenStreetMap.org Web site. Once uploaded, everybody can annotate, categorize and correct the data. The data structures to capture this information are points and ways to connect these points. By annotation or tagging points, the project in particular produces a vast amount of point-of-interest descriptions – hotels, gas stations, traffic lights, shops, base transceiver stations, to name a few. Overall, the project in particular produces a vast amount of point-of-interest descriptions – hotels, gas stations, traffic lights, ambulance, amusement park, art gallery, atm, pub, church

<table>
<thead>
<tr>
<th>Property</th>
<th>Usage</th>
<th>Values</th>
<th>Value examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>1.066k</td>
<td>704k</td>
<td>airport, attraction, building, continent, island, village, city</td>
</tr>
<tr>
<td>place</td>
<td>443k</td>
<td>287</td>
<td>airline office, airport, alm hut, church</td>
</tr>
<tr>
<td>amenity</td>
<td>277k</td>
<td>1433</td>
<td>attraction, building</td>
</tr>
<tr>
<td>ia:in</td>
<td>190k</td>
<td>14k</td>
<td>access, bench, bridge, bus stop, traffic signals</td>
</tr>
<tr>
<td>highway</td>
<td>188k</td>
<td>310</td>
<td>station, bridge, construction, junction</td>
</tr>
<tr>
<td>postal_code</td>
<td>140k</td>
<td>63.135</td>
<td>bay, beach, bog, cairn, caldera, cave, caves, cave, entrance, cliff, coastline</td>
</tr>
<tr>
<td>railway</td>
<td>53k</td>
<td>111</td>
<td>dry cleaning, patisserie, supermarket, technology, airline</td>
</tr>
<tr>
<td>natural</td>
<td>28k</td>
<td>149</td>
<td>accommodation, alpine hut, apartments, art gallery, attraction, bar, boat rental</td>
</tr>
<tr>
<td>tourism</td>
<td>27k</td>
<td>519</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2:** Examples of frequently used properties and values in the OpenStreetMap data.

The resulting database schema can be easily published by using Triplify. However, in order to retrieve information, the point or way identifiers (i.e. primary keys from the respective columns) have to be known, which is usually not the case. A natural entrypoint for retrieving geo data, however, is the neighborhood around a particular point, possibly filtered by points adhering to certain categories or being of a certain type. To support this usage scenario, we developed a spatial Linked Data extension, which allows to retrieve geo data of a particular circular region. The structure of the URIs used looks as follows:

```
```

The linked geo data extension is implemented in Triplify by using a configuration with regular expression URL patterns which extract the geo coordinates, radius and optionally a property with associated value and insert this information into an SQL query for retrieving corresponding points of interest. The SQL query is optimized so as to retrieve first points in the smallest rectangle covering the requested circular area and then cutting the result set into a circular area by using the Haversine Formula. Performance results for retrieving points-of-interest in different areas are summarized in Table 3.

The publication of the OpenStreetMap data using Triplify adds a completely new dimension to the Data Web: spatial data can be retrieved and interlinked on an unprecedented level of granularity. This enhancement enables a variety of new Linked Data applications such as geo data syndication or semantic-spatial searches. The dynamic of the OpenStreetMap project will ensure a steady growth of the dataset. Usage instructions and further information can be also found at [http://LinkedGeoData.org](http://LinkedGeoData.org).
four classes of different approaches: the database semantics as starting point are Tirmizi [16], of alignment quality also remains an open issue. Evaluation need for an existing domain ontology which has to be defined a priori and independently from the database. Queries are only allowed based on forms that are generated from manually align the database to an existing ontology. Queries can be found in [10] and [14].

6. RELATED WORK

The amount of work related to transformations between relational and RDF data models is extensive and an exhaustive overview cannot be given here. Nevertheless, we will give an overview of the major approaches. The W3C RDB2RDF Incubator Group\(^6\) has taken an effort to classify existing approaches\(^6\). Other overviews and classifications can be found in [10] and [14]\(^7\).

The following classification criteria can be identified:

(a) Degree of mapping creation automation (automatic, semi-automatic, manual approaches).

(b) Some approaches are tailored to model a domain, sometimes with the help of existing ontologies, while others attempt to extract domain information primarily from the given database schema with few other resources used (domain or database semantics-driven). The latter often results in a table-to-class, column-to-predicate mapping. Some approaches also use a (semi) automatic approach based on the database, but allow manual customization to model domain semantics.

(c) Resulting access paradigm (ETL, Linked Data, SPARQL access). Note that the access paradigm also determines whether the resulting RDF model updates automatically. ETL means a one time conversion, while Linked Data and SPARQL always process queries versus the original database.

(d) The used mapping language as an important factor for reusability and initial learning cost.

(e) Domain reliance (general or domain-dependent): requiring a pre-defined ontology is a clear indicator of domain dependency.

A general overview is given in Table 4. We have identified four classes of different approaches:

Alignment. Dartgrid [17] uses a visual mapping tool to manually align the database to an existing ontology. Queries are only allowed based on forms that are generated from the ontology. Hu [11] is one of the latest representatives of this class of approaches where databases are automatically aligned to a reference ontology. The major drawback is the need for an existing domain ontology which has to be defined a priori and independently from the database. Evaluation of alignment quality also remains an open issue.

Database Mining. Four similar approaches which use the database semantics as starting point are Tirmizi [16],

<table>
<thead>
<tr>
<th>Location</th>
<th>Radius</th>
<th>Property</th>
<th>Results</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leipzig</td>
<td>1km</td>
<td>-</td>
<td>291</td>
<td>0.05s</td>
</tr>
<tr>
<td>Leipzig</td>
<td>5km</td>
<td>amenity=pub</td>
<td>41</td>
<td>0.54s</td>
</tr>
<tr>
<td>London</td>
<td>1km</td>
<td>-</td>
<td>259</td>
<td>0.28s</td>
</tr>
<tr>
<td>London</td>
<td>5km</td>
<td>amenity=pub</td>
<td>495</td>
<td>0.74s</td>
</tr>
<tr>
<td>Amsterdam</td>
<td>1km</td>
<td>-</td>
<td>1811</td>
<td>0.31s</td>
</tr>
<tr>
<td>Amsterdam</td>
<td>5km</td>
<td>amenity=pub</td>
<td>64</td>
<td>1.25s</td>
</tr>
</tbody>
</table>

Table 3: Performance results for retrieving points-of-interest in different areas.

7. CONCLUSIONS

Turning the Semantic Data Web into reality still poses a significant challenge, although standards, technologies, and tools are already in place. In order to overcome the chicken-and-egg problem between semantic representations and search facilities on the Web, we developed the Triplify approach which primarily tackles the ‘reaching a critical mass of semantics’ side of the problem. We hope that Triplify will contribute a strong impulse for achieving a larger deployment of semantic technologies on the Web. Triplify provides direct benefits to the developers and users of a Web application:

- Installations of Web applications are easier to find and search engines can better evaluate their data.
- The usage of Triplify is minimally invasive by only requiring few application changes. Approaches which integrate data representations into the HTML output (such as RDFa and GRDDL), in contrast, require larger modifications.
- Different installations of the same Web application can easily syndicate arbitrary content without the need to adopt interfaces, content representations or protocols.
- Triplify facilitates the creation of custom-tailored search engines targeted at certain niches, e.g. searching for specific content in various blogs, wikis, or forums.

\(7\) http://esw.w3.org/topic/Rdb2RdfXG/StateOfTheArt
\(^16\) http://www.w3.org/2005/Incubator/rdb2rdf/

Li [12], DB2OWL [10], and RDBToOnto [6]. Tirmizi has a formal system to capture the complete information contained in the database which is based on the idea that all domain semantics is already contained in the database. Li, DB2OWL and RDBToOnto use less complete extraction rules. Li and RDBToOnto also aim at refining the resulting ontology. RDBToOnto provides a visual interface for manual changes. DB2OWL creates a local ontology from a database which is later aligned to a reference ontology. The ontologies created in these approaches reflect the database semantics. Problems are likely to arise, when several different databases need to be integrated due to the lack of database-independent domain semantics.

Integration. A typical integration project is described in [15]. A custom-tailored mapping is created on the basis of XSLT. Although the project delivers no technical methods for mapping, it clearly shows how much expertise is needed from domain experts to capture semantics correctly.

Languages/Servers. R2O [13] and D2RQ [4] are mapping languages, but follow different goals. While R2O provides more flexibility and expressiveness, it also needs a reference ontology. D2RQ directly exposes the database as Linked Data and SPARQL with D2R server. Both mapping languages require an initial learning of the language as well as knowledge about modelling. Virtuoso RDF Views [5, 9] stores mappings in a quad storage. While D2RQ and RDF Views follow the table-to-class, column-to-predicate approach, RDF Views has some more methods to incorporate DB semantics. None of the approaches provide a methodology for users as such, but rather give an ontological representation of the database. The mapping languages used have about the same complexity and flexibility as SQL. (SQL has more flexible selection capabilities, while the languages concentrate on expressing the ontology mapping.)
Ultimately, a more widespread use of Data Web technologies will counteract the centralization we faced through the erection of huge data silos of the Web 2.0 and will lead to an increased democratization of the Web.

**Future Work.** Triplify deliberately does not support SPARQL queries. Adding SPARQL support to Triplify would raise questions about security and scalability among Web application developers and administrators. On the other hand, we think that employing SQL as a mapping language to RDF and ontologies is a very powerful approach and it will be hard for newly developed mapping languages to achieve similar flexibility. Hence, we envision some extensions to Triplify such as a more external annotation of the SQL views in order to allow optionally SPARQL processing on Triplify endpoints. We also aim at improving the OpenStreetMap data usage scenario, e.g. by better interlinking the data with other Linked Data datasets and providing a proper ontology for querying.

### 8. ACKNOWLEDGMENTS

We thank Elias Theodorou for contributing to the Triplify integrations. We thank the members of the W3C R2RDF XG for numerous discussions – in particular Satya Sahoo and Wolfgang Halb for editing the literature survey, providing valuable input for the related work section.

### 9. REFERENCES


### Table 4: An integrated overview of mapping approaches. Criteria for classification were merged, some removed, fields were completed, when missing. DB+M means that the semi-automatic approach can later be customised manually

<table>
<thead>
<tr>
<th>Approach</th>
<th>Automation</th>
<th>Domain or database semantics-driven</th>
<th>Access paradigm</th>
<th>Mapping language</th>
<th>Domain reliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hu et al. [11]</td>
<td>Auto</td>
<td>Both</td>
<td>ETL</td>
<td>intern</td>
<td>dependent</td>
</tr>
<tr>
<td>Tirmizi et al. [16]</td>
<td>Auto</td>
<td>DB</td>
<td>ETL</td>
<td>FOL</td>
<td>general</td>
</tr>
<tr>
<td>Li et al. [12]</td>
<td>Semi</td>
<td>DB</td>
<td>ETL</td>
<td>n/a</td>
<td>general</td>
</tr>
<tr>
<td>DB2OWL[10]</td>
<td>Semi</td>
<td>DB</td>
<td>SPARQL</td>
<td>R2O</td>
<td>general</td>
</tr>
<tr>
<td>D2RQ[4]</td>
<td>Auto</td>
<td>DB+M</td>
<td>LD, SPARQL</td>
<td>D2RQ</td>
<td>general</td>
</tr>
<tr>
<td>Virtuoso RDF View [5, 9]</td>
<td>Semi</td>
<td>DB+M</td>
<td>SPARQL</td>
<td>own</td>
<td>general</td>
</tr>
<tr>
<td>Triplify</td>
<td>Manual</td>
<td>Domain</td>
<td>LD</td>
<td>SQL</td>
<td>general</td>
</tr>
</tbody>
</table>

*Table 4: An integrated overview of mapping approaches. Criteria for classification were merged, some removed, fields were completed, when missing. DB+M means that the semi-automatic approach can later be customised manually.*