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

WWW 2009, April 20–24, 2009, Madrid, Spain.
ACM 978-1-60558-487-4/09/04.

Project	**Area**	**Downloads**
phpBB	discussion forum	170,484
Gallery	photo gallery	115,966
Liferay Portal	Portal	82,442
Coppermine	photo gallery	76,702
XOOPS	CMS	69,404
Typo3	CMS	58,651
eGroupWare	group ware	28,370
PHP-Fusion	CMS	27,090
Alfresco	CMS	19,851
e107	CMS	19,420
Lifetype	Blogging	16,867
WebCalendar	Calendar	11,776
Compiere	ERP + CRM	11,522
Tikiwiki	Wiki	10,092
Nucleus	Blogging	5,490

Table 1: The 15 most popular database-backed web application projects as hosted at Sourceforge.net and ordered by download count in September 2008.

of generating mappings. Tools which automate the generation of mappings fail frequently in creating production-ready mappings. Obstacles, for example, include:

- *Identification of private and public data.* Web applications always contain information which should not be made public on the Web such as passwords, email addresses or technical parameters and configurations. Automatically distinguishing between strictly confidential, important and less relevant information is very hard, if not impossible.

- *Proper reuse of existing vocabularies.* Even the most elaborated approaches to ontology mapping fail in generating certain mappings between the database entities (table and column names) and existing RDF vocabularies, due to lacking machine-readable descriptions of the domain semantics in the database schema.

- *Missing database schema descriptions.* Many database schemas used in Web applications lack comprehensive definitions for foreign keys or constraints, for example. Syntactic approaches for detecting these are likely to fail, since database schemas are often grown evolutionary and naming conventions are often not enforced.

Taking these obstacles into account, the entrance barrier for publishing database content as RDF appears to be very high. On the other hand, Web applications are often deployed a thousand (if not a million) times on the Web (cf. Table 1) and database schemas of Web applications are significantly simpler than those of ERP systems, for instance. Likewise, Web application developers and administrators usually have decent knowledge of relational database technologies, in particular of SQL.

The rationale behind Triplify is to provide for a specifically tailored RDB-to-RDF mapping solution for Web applications. The ultimate aim of the approach is to lower the entrance barrier for Web application developers as much as possible. In order to do so, Triplify neither defines nor requires to use a new mapping language, but exploits and extends certain SQL notions with suitable conventions for transforming database query results (or views) into RDF and Linked Data.

Once a large number of relational data is published on the Web, the problem of tracing updates of this data becomes paramount for crawlers and indexes. Hence, Triplify is complemented by an approach for publishing update logs of relational databases as Linked Data, essentially employing the same mechanisms. Triplify is also accompanied with a repository for Triplify configurations in order to facilitate reuse and deployment, and with a light-weight registry for Triplify and Linked Data endpoints. We tested and evaluated Triplify by integrating it into a number of popular Web applications. Additionally, we employed Triplify to publish the 160GB of geo data collected by the OpenStreetMap project.

The paper is structured as follows: We motivate the need for a simple RDB-to-RDF mapping solution in Section 2 by comparing indicators for the growth of the Semantic Web with those for the Web. We present the technical concept behind Triplify in Section 3. Our implementation is introduced in Section 4. We discuss the evaluation of Triplify in small- and large-scale scenarios in Section 5, present a comprehensive overview on related work in Section 6, and conclude with an outlook on future work in Section 7.

2. CURRENT GROWTH OF THE SEMANTIC WEB

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 engines index from known or computed word frequencies of common words[1]. Along the line of similar studies, the statistics suggest an exponential growth of pages on the WWW. The number of hosts[2] (green line) grows slower, while still being roughly exponential[3].

We tried to relate this to the growth of the Semantic Web. The currently most complete index of Semantic Web data is probably Sindice[4]. 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[5]. The blue

[1] see *http://www.worldwidewebsize.com*

[2] statistics from *http://isc.org*

[3] see *http://www.isc.org/ops/ds/hosts.png* for a long term analysis

[4] *http://sindice.com*

[5] *http://swoogle.umbc.edu/*

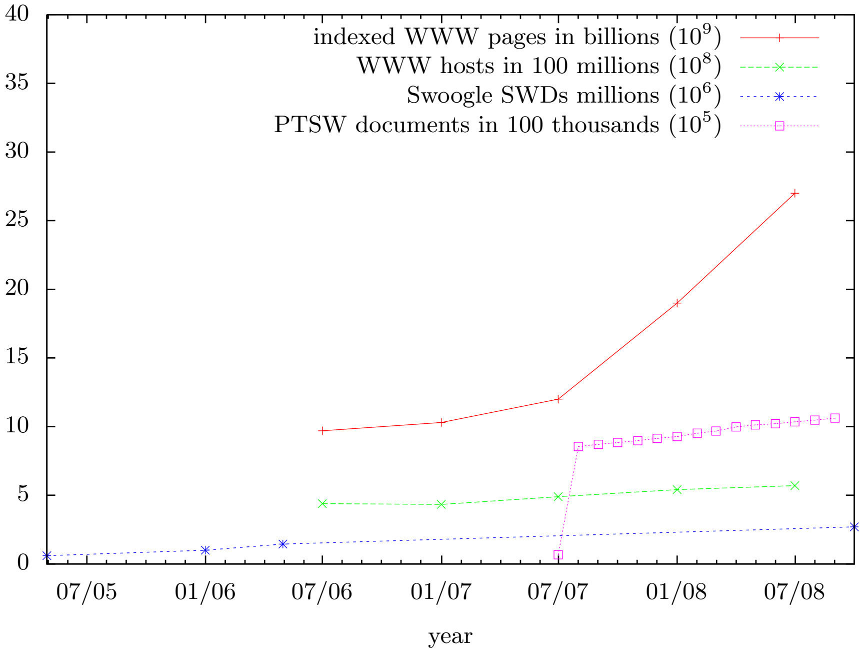


Figure 1: Temporal comparison of different growth indicators for the traditional WWW and the Semantic Data Web. The first and second lines show statistics for indexed web pages (obtained from WorldWideWebSize.com) and hosts (obtained from isc.org) for the WWW. The third and fourth lines are counts of 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)[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 [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[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, [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, ϕ, μ) where:

- s is a default schema namespace,
- ϕ is a mapping of namespace prefixes to namespace URIs,

[6] *http://pingthesemanticweb.com/*
[7] *http://iws.seu.edu.cn/services/falcons/*

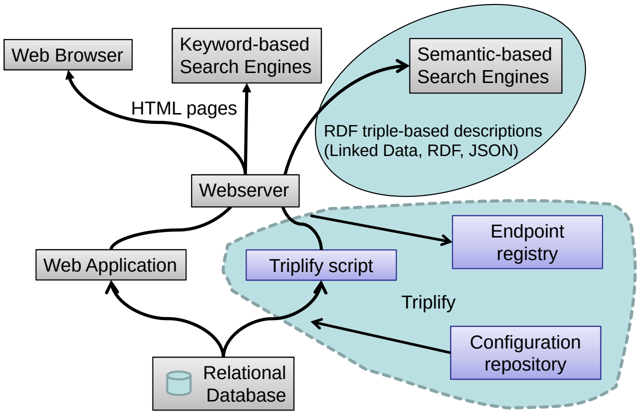


Figure 2: Triplify overview: the Triplify script is accompanied with a configuration repository and an endpoint registry.

- μ is a mapping of URL patterns to sets of SQL queries; individual queries can optionally contain placeholders referring to parenthesized sub-patterns in the corresponding URL pattern.

The default schema namespace s is used for creating URI identifiers for database view columns, which are not mapped to existing vocabularies. Since the same Triplify configuration can be used for all installations of Web applications with the same database storage layout, data from across these different installations can be already integrated even without a mapping to existing vocabularies. The namespace prefix to URI map ϕ in the Triplify configuration defines shortcuts for frequently used namespaces. The mapping μ maps dereferencable RDF resources to sets of SQL views describing these resources. The patterns in the domain of μ will be evaluated against URI requests Triplify receives via HTTP. The Triplify extraction algorithm replaces placeholders in the SQL queries with matches of parenthesized sub-patterns in the URL patterns of μ. For the purpose of simplicity, our implementation (which we discuss in more detail later) does not require the use of patterns and placeholders at all, and is still sufficient for the vast majority of usage scenarios. A simple example of μ for the WordPress blogging system is displayed in Figure 3.

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:
  `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).
- *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 '`^^`'.
- *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:

```
$triplify['queries']=array(
  'post'=>array(
    "SELECT id, post_author AS 'sioc:has_creator->user', post_date AS 'dcterms:created', post_title AS 'dc:title',
        post_content AS 'sioc:content', post_modified AS 'dcterms:modified'
      FROM posts WHERE post_status='publish'",
    "SELECT post_id AS id, tag_id AS 'tag:taggedWithTag->tag' FROM post2tag",
    "SELECT post_id AS id, category_id AS 'belongsToCategory->category' FROM post2cat",
  ),
  'tag'=>"SELECT tag_ID AS id, tag AS 'tag:tagName' FROM tags",
  'category'=>"SELECT cat_ID AS id, cat_name AS 'skos:prefLabel', category_parent AS 'skos:narrower->category'
      FROM categories",
  'user'=>array(
    "SELECT id, user_login AS 'foaf:accountName', user_url AS 'foaf:homepage',
        SHA(CONCAT('mailto:',user_email)) AS 'foaf:mbox_sha1sum', display_name AS 'foaf:name'
      FROM users",
    "SELECT user_id AS id, meta_value AS 'foaf:firstName' FROM usermeta WHERE meta_key='first_name'",
    "SELECT user_id AS id, meta_value AS 'foaf:family_name' FROM usermeta WHERE meta_key='last_name'",
  ),
  'comment'=>"SELECT comment_ID AS id, comment_post_id AS 'sioc:reply_of->user', comment_author AS 'foaf:name',
      comment_author_url AS 'foaf:homepage', SHA(CONCAT('mailto:',comment_author_email)) AS 'foaf:mbox_sha1sum',
      comment_type, comment_date AS  'dcterms:created', comment_content AS 'sioc:content', comment_karma,
    FROM comments WHERE comment_approved='1'",
);
```

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 URL, replaces potential placeholders in the associated SQL queries with matching parts in the request URL, issues the queries and transforms the returned results into RDF (cf. Algorithm 1). The processing steps of the algorithm are depicted in Figure 4, using the example of the WordPress Triplify configuration from Figure 3.

Linked Data Generation. The Linked Data paradigm is based on the idea of making URIs used in RDF documents de-referencable – i.e. accessible via the HTTP protocol. The result of such an HTTP request delivers a description of the resource identified by the URI, i.e. a collection of all relevant information about the resource. The Linked Data paradigm of publishing RDF solves several important issues: (a) the provenance of facts expressed in RDF can be easily verified, since the used URIs always contain authoritative information in terms of the publishing server's domain name, (b) the (continuing) validity of facts can be easily verified by (re-)retrieving the RDF description, (c) Web crawlers can obtain information in small chunks and follow RDF links to gather additional, linked information.

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 re-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,

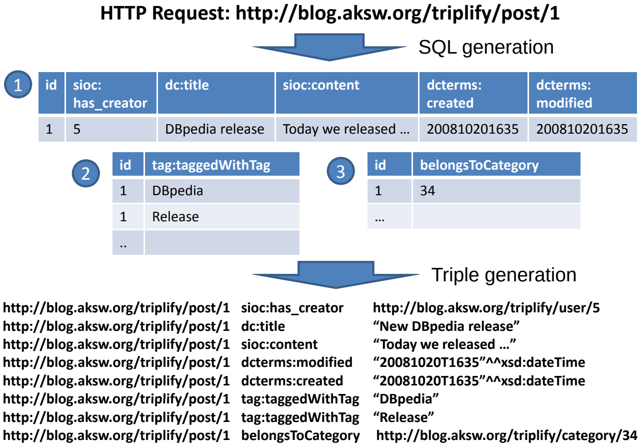
[8] *http://pingthesemanticweb.com/*

Algorithm 1: Triple Extraction Algorithm

```
Input: request URL, Triplify configuration
Output: set $T$ of triples
foreach URL pattern from Triplify configuration do
    if request URL represents endpoint request then
        $T = T \cup$ {RDF link to class request for URL pattern}
    else
        if request URL matches URL pattern then
            foreach SQL query template associated with URL pattern do
                $Query$ = replacePlaceholder($URL pattern, SQL query template, request URL$);
                if request URL represents class request then
                    $Query$ = projectToFirstColumn ($Query$);
                else
                    if request URL represents instance request then
                        $Query$ = addWhereClause ($Query, instanceId$);
                $Result$ = execute($Query$);
                $T = T \cup$ convert($Result$);
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.



id	sioc: has_creator	dc:title	sioc:content	dcterms: created	dcterms: modified
1	5	DBpedia release	Today we released …	200810201635	200810201635

id	tag:taggedWithTag
1	DBpedia
1	Release
..	

id	belongsToCategory
1	34
…	

http://blog.aksw.org/triplify/post/1	sioc:has_creator	http://blog.aksw.org/triplify/user/5
http://blog.aksw.org/triplify/post/1	dc:title	"New DBpedia release"
http://blog.aksw.org/triplify/post/1	sioc:content	"Today we released …"
http://blog.aksw.org/triplify/post/1	dcterms:modified	"20081020T1635"^^xsd:dateTime
http://blog.aksw.org/triplify/post/1	dcterms:created	"20081020T1635"^^xsd:dateTime
http://blog.aksw.org/triplify/post/1	tag:taggedWithTag	"DBpedia"
http://blog.aksw.org/triplify/post/1	tag:taggedWithTag	"Release"
http://blog.aksw.org/triplify/post/1	belongsToCategory	http://blog.aksw.org/triplify/category/34

Figure 4: Triple generation from database views. The numbered query result relations correspond to the queries defined for the post URL pattern in the WordPress example configuration in Figure 3.

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`, for example, will return the following RDF:

```
http://myBlog.de/lod/update/2007
                    rdf:type   update:UpdateCollection .
http://myBlog.de/lod/update/2008
                    rdf:type   update:UpdateCollection .
```

Each update collection should be additionally annotated with the timeframe it covers (which we omit here for reasons of brevity), to avoid the need for an interpretation of the URI structure. The following RDF could then be returned for `http://example.com/lod/update/2008`:

```
http://myBlog.de/lod/update/2008/Jan
                    rdf:type   update:UpdateCollection .
http://myBlog.de/lod/update/2008/Feb
                    rdf:type   update:UpdateCollection .
```

This nesting continues until we finally reach an URL which exposes all blog updates performed in a certain second in time. The resource `http://myBlog.de/lod/update/2008/Jan/01/17/58/06` then would, for example, contain RDF links (and additional metadata) to the Linked Data documents updated on Jan 1st, 2008 at 17:58:06:

```
http://myBlog.de/lod/update/2008/Jan/01/17/58/06/user123
  update:updatedResource http://myBlog.de/lod/user/John;
  update:updatedAt   "20080101T17:58:06"^<xsd:dateTime>;
  update:updatedBy   http://myBlog.de/lod/user/John .
```

Invocation. Triplify automatically generates all the resources in the update URI space, when the mapping μ in the Triplify configuration contains the URL pattern "update". Queries belonging to this URL pattern have to return at least two columns. The first column contains the date when to update occurred, the second column contains the id of the updated resource. Additional columns can contain additional metadata. An example query publishing updates of a WordPress blog is given below:

```
SELECT post_modified,id AS 'update:updatedResource->post'
FROM post
```

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 URIs. When SQL functions are insufficient or inconvinient 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 preceeding 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*

tology[13], but complemented with additional attributes that are defined in a separate Triplify osCommerce 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:Usergoup`) 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[14]. 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 had produced a 160GB database of geo data until July 2008, in some regions surpassing commercial geo data providers in terms of precision and detail.

In order to publish the OpenStreetMap data, we performed some preprocessing of the data structures. Point annotations, for example, are originally stored as comma separated property-values assignments in a BLOB column within the database. These were decoded and stored in a separate table. We also created a stripped-down version of the points' table, containing only points with annotations and eliminating those which are just used as hanging points for ways. As a result, we obtained 192 million points-of-interest, which are annotated with roughly 800 million property-value combinations. Table 2 summarizes the most popular point-of-interest annotations currently found in the OpenStreetMap data. Overall, our OpenStreetMap data triplification resulted in roughly 1 billion triples constituting one of the largest available datasets on the Semantic Web and by far surpassing the DBpedia [1] effort.

Property	Usage	Values	Value examples
name	1.066k	704k	
place	443k	287	airport, attraction, building, continent, island, village, city
amenity	277k	1433	airline office, airport, alm hut, ambulance, amusement park, art gallery, atm, pub, church
is_in	190k	14k	
highway	188k	310	access, bench, bridge, bus stop, traffic signals
postal_code	140k	63.135	
railway	53k	111	station, bridge, construction, junction
natural	28k	149	bay, beach, bog, cairn, caldera, cave, caves, cave_entrance, cliff, coastline
shop	27k	803	dry cleaning, patisserie, supermarket, technology, airline
tourism	27k	519	accommodation, alpine hut, apartments, art gallery, attraction, bar, boat rental

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:

$$\text{http://LinkedGeoData.org/near/}\overbrace{\text{48.213}}^{\text{Longitude}}\text{,}\overbrace{\text{16.359}}^{\text{Latitude}}\text{/}\overbrace{\text{1000}}^{\text{Radius}}\text{/}\overbrace{\text{amenity=pub}}^{\text{Property}}$$

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.*

[13] *http://www.heppnetz.de/projects/goodrelations/*

[14] *http://www.openstreetmap.org*

Location	Radius	Property	Results	Time
Leipzig	1km	-	291	0.05s
Leipzig	5km	amenity=pub	41	0.54s
London	1km	-	259	0.28s
London	5km	amenity=pub	495	0.74s
Amsterdam	1km	-	1811	0.31s
Amsterdam	5km	amenity=pub	64	1.25s

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

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[15] has taken an effort to classify existing approaches[16]. Other overviews and classifications can be found in [10] and [14][17].

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 representants 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], 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.)

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.

[15] *http://www.w3.org/2005/Incubator/rdb2rdf/*
[16] *http://esw.w3.org/topic/Rdb2RdfXG/StateOfTheArt*
[17] *http://www2006.org/programme/files/pdf/p160-slides.pdf*

Approach	**Automation** (a)	**Domain or database semantics-driven** (b)	**Access paradigm** (c)	**Mapping language** (d)	**Domain reliance** (e)
Dartgrid [17]	Manual	Domain	SPARQL	Visual Tool	dependent
Hu et al. [11]	Auto	Both	ETL	intern	dependent
Tirmizi et al. [16]	Auto	DB	ETL	FOL	general
Li et al. [12]	Semi	DB	ETL	n/a	general
DB2OWL[10]	Semi	DB	SPARQL	R2O	general/dependent
RDBToOnto [6]	Semi	DB+M	ETL	Visual Tool	general
Sahoo et al. [15]	Manual	Domain	ETL	XSLT	dependent
R2O[13]	Manual	DB+M	SPARQL	R2O	dependent
D2RQ[4]	Auto	DB+M	LD, SPARQL	D2RQ	general
Virtuoso RDF View [5, 9]	Semi	DB+M	SPARQL	own	general
Triplify	Manual	Domain	LD	SQL	general

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 customized manually

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 RDB2RDF 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

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. DBpedia: A nucleus for a web of open data. In *ISWC/ASWC (2007)*, LNCS (4825), pages 722–735. Springer, 2007.

[2] T. Berners-Lee. Design issues: Linked data, 2006. http://www.w3.org/DesignIssues/LinkedData.html.

[3] C. Bizer, R. Cyganiak, and T. Heath. How to publish linked data on the web, 2007. http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/.

[4] C. Bizer and A. Seaborne. D2RQ - treating non-RDF databases as virtual RDF graphs. In *ISWC2004 (posters)*, November 2004.

[5] C. Blakeley. Rdf views of sql data (declarative sql schema to rdf mapping), 2007.

[6] F. Cerbah. Learning highly structured semantic repositories from relational databases. In *ESWC*, volume 5021 of *LNCS*, pages 777–781. Springer, 2008.

[7] G. Cheng and Y. Qu. Term dependance on the semantic web. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2008.

[8] L. Ding and T. Finin. Characterizing the semantic web on the web. In *Proceedings of ISWC 2006, Athens, GA, USA, November 5-9, 2006*, volume 4273 of *LNCS*, pages 242–257. Springer, 2006.

[9] O. Erling and I. Mikhailov. RDF support in the Virtuoso DBMS. In *Proceedings of the 1st Conference on Social Semantic Web*, volume P-113 of *GI-Edition - Lecture Notes in Informatics (LNI), ISSN 1617-5468*. Bonner Köllen Verlag, September 2007.

[10] R. Ghawi and N. Cullot. Database-to-ontology mapping generation for semantic interoperability, 2007. Third International Workshop on Database Interoperability (InterDB), 2007

[11] W. Hu and Y. Qu. Discovering simple mappings between relational database schemas and ontologies. In *Proceedings of ISWC/ASWC2007, Busan, South Korea*, volume 4825 of *LNCS*, pages 225–238, 2007

[12] M. Li, X. Du, and S. Wang. A semi-automatic ontology acquisition method for the semantic web. In W. Fan, Z. Wu, and J. Yang, editors, *WAIM*, volume 3739 of *LNCS*, pages 209–220. Springer, 2005.

[13] J. B. Rodríguez, Óscar Corcho, and A. Gómez-Pérez. R2o, an extensible and semantically based database-to-ontology mapping language. SWDB, 2004.

[14] J. B. Rodríguez and A. Gómez-Pérez. Upgrading relational legacy data to the semantic web. In L. Carr, D. D. Roure, A. Iyengar, C. A. Goble, and M. Dahlin, editors, *WWW*, pages 1069–1070. ACM, 2006.

[15] S. S. Sahoo, O. Bodenreider, J. L. Rutter, K. J. Skinner, and A. P. Sheth. An ontology-driven semantic mashup of gene and biological pathway information: Application to the domain of nicotine dependence. *Journal of biomedical informatics*, February 2008.

[16] S. H. Tirmizi, J. Sequeda, and D. P. Miranker. Translating sql applications to the semantic web. In *DEXA*, volume 5181 of *LNCS*, pages 450–464. 2008

[17] Z. Wu, H. Chen, H. Wang, Y. Wang, Y. Mao, J. Tang, and C. Zhou. Dartgrid: a semantic web toolkit for integrating heterogeneous relational databases. In *Semantic Web Challenge at 4th International Semantic Web Conference*, Athens, USA, NOV 2006.