Linking Semistructured Data on the Web

Soheil Hassas Yeganeh
University of Toronto
soheil@cs.toronto.edu

Oktie Hassanzadeh
University of Toronto
oktie@cs.toronto.edu

Renée J. Miller
University of Toronto
miller@cs.toronto.edu

ABSTRACT

Many Web data sources and APIs make their data available in XML, JSON, or a domain-specific semi-structured format, with the goal of making the data easily accessible and usable by Web application developers. Although such data formats are more machine-processable than pure text documents, managing and analyzing such data in large scale is often nontrivial. This is mainly due to the lack of a well-defined (or understood) structure and clear semantics in such data formats, which could result in poor data quality. In the xCurator project, we add structure to such data with the goal of publishing it on the Web as Linked Data. We enhance the quality of such data by: extracting entities, their types, and their relationships to other entities; performing entity (and entity type) identification; merging duplicate entities (and entity types); linking related entities (internally and to external sources); and publishing the results on the Web as high-quality Linked Data. This is all in a light-weight easy-to-use and scalable framework that effectively incorporates user feedback in all phases. We describe the initial framework of our system and report the results of using our system for managing large volumes of (user-generated) data on the Web in several real world applications.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems; H.2.5 [**Database Management**]: Heterogeneous Databases

Keywords

Semistructured data, Schema Discovery, Linked Data

1. INTRODUCTION

Over the past decade, there has been a massive increase in the amount of semistructured data on the Web. Data and Web service providers often use generic data formats such as JSON, XML or XML-based formats, or other industry-standard or domain-specific text-based formats such as BibTeX (and other bibliographic data formats), XMCD (audio CD data format) or DrugCard (drug information) to name a few. Although using such data formats has made data sharing and exchange significantly easier, the management, analysis, and querying of such data has become increasingly difficult. This is mainly due to the lack of (an enforced) *schema* and clear *semantics* in such formats. As a result, both the data values and their structure may be inconsistent and contain errors, making it difficult for users to understand and query the data source contents.

On the other hand, many tools, systems, technologies and standards have been developed in order to realize the vision of the Semantic Web, a *Web of Data* consisting of *objects* (or *entities*) with facts (or *triples*) that describe their relationships and attributes in a reasonably structured way, using the Resource Description Framework (RDF) data model. In particular, there has been an increasing interest in following the *Linked Data* principles to publish data on the Web, as a part of the Linking Open Data (LOD) community project at W3C [10]. These principles significantly enhance adaptability and usability of the data on the Web. The number of triples published by the LOD data sources (aka *LOD cloud*) has grown from around 500 million triples in 2007, to more than 25 billion triples that currently describe millions of interlinked entities in various domains. Part of the success of the LOD project relies on tools that assist data publishers in publishing Linked Data out of existing structured sources. In particular, a class of tools known as RDB2RDF systems [19] have been developed that transform existing relational sources into RDF or provide dynamic RDF views. However, very few tools exist that assist data publishers in exposing semistructured data as Linked Data on the Web in a similarly generic way. This is mainly due to the inherent difficulty of such transformations, difficulties that go well beyond data model translation (translating XML data to RDF data) or schema inference.

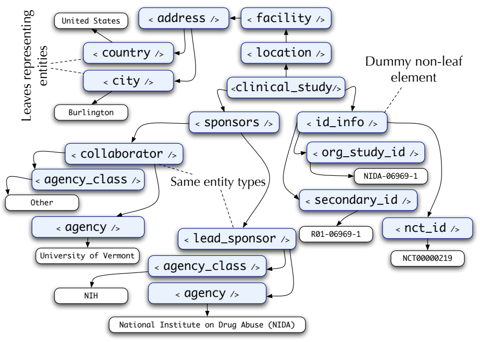


Figure 1: Sample XML elements from Clinical Trials

Consider the XML data shown in Figure 1 describing a clinical trial in ClinicalTrials.gov. Transforming this data into high-quality Linked Data involves multiple steps. First, we need to perform *entity type identification*, which involves detecting entities (or *resources*), their types (or *classes*) and attributes (or *properties*). This information can be derived from a reasonable sample of data. A simple heuristic might consider non-leaf elements as entity types (e.g., `<clinical_study/>`), while leaf elements containing only text (e.g., `<agency_class/>`) are translated into attributes of the parent entity type. However, simple heuristics like this will not always lead to high-quality *linkable* data. For instance, in Figure 1, there is a non-leaf, container element (`<id_info/>`)

which does not represent an entity type, and there is a leaf element (`<country/>`) which is an entity type. It is important to ensure country values are represented as entities (not literal values) and can be linked to appropriate entities in the LOD cloud. Another challenge associated with automatic entity type identification is detecting *duplicate types*. In Figure 1, there are two identical subtrees in the structure (`<lead_sponsor/>` and `<collaborator/>`). These subtrees are different instances of the same entity type, which may not be obvious at first glance. Such subtrees should be detected as identical types in order to avoid data duplication, and to make it possible to properly link related entities.

After detecting entity types and their attributes, the data needs to be transformed into triples that describe the entities, their relationships, and their attributes. A major challenge in this transformation is the existence of duplicate instances. Such duplicates may result from (*i*) the existence of multiple occurrences of the same entity in different locations in a data set, (*ii*) different versions of the same data, and (*iii*) actual duplicates in the source. The problem is further complicated by the existence of *fuzzy duplicates*, i.e., entities that have different representations, but refer to the same real-world entity. For example, once we identify `<country/>` as an entity type, we need to detect whether two instances with different representations (e.g., "United States" and "U.S.A") refer to the same country and, if so, merge them into a single instance (entity).

The final step in this process is *interlinking entities* and their types to external knowledge repositories and data sources. There could be several ontologies and Web sources that contain information about our source entities and their types. Linking from entity types to external ontologies can enhance entity type identification and enhance the quality of the identified types, in addition to the benefit to entity identification. As an example, by matching leaf element `<country/>` with Freebase's `/location/country` or DBpedia's `dbpedia-owl:Country` entity types, we find additional evidence for our identified type, and we will be able to link the entity with label "U.S.A." with Freebase resource `/en/united_states` or DBpedia resource `dbpedia:United_States`.

Contributions. In this paper, we present xCurator, a system capable of transforming semistructured data into high-quality Linked Data automatically with *optional* human intervention. Our contribution is threefold:

• We present an end-to-end framework for transforming a possibly large set of semistructured data instances into rich high-quality Linked Data. The input to the system can be instances of static semistructured data sources available on the Web, or dynamic user-generated content such as BibTeX entries, or RSS feeds.

• We present a brief overview of our implementation of each component of the proposed framework, and more detailed discussion of the entity type extraction process. We use or extend existing techniques from the data management literature to address the above-mentioned challenges.

• We report the results of applying our proposed framework to transforming several real-world data sets from different domains into rich Linked Data. The results include a bibliographic data source generated from online user-generated BibTeX files, and a data source of clinical trials generated from thousands of online XML descriptions. Part of the results have been made available on the Web as a part of the Linked Open Data cloud.

In the following section, we present the architecture of our framework and a brief overview of related work for each component of the framework. Section 3 presents a detailed description of the entity type extraction process. We report the results of using our system in several real-world scenarios in Section 4. Section 5 concludes the paper and presents a few interesting future directions.

2. FRAMEWORK

Figure 2 shows the xCurator framework. The input to this framework is a semistructured data source, with no restriction on the characteristics of the data. For example, the data could be a single huge XML file stored locally containing all the DBLP publications, or can be URLs of millions of small online BibTeX files, each file containing one or more publication entries. The data could be static or dynamic, i.e., the data and its structure could change at any time. The output of the system is *high-quality* Linked Data, meaning: (*i*) objects that are identified by unique HTTP URIs; (*ii*) when objects are looked up, RDF statements are returned describing the object; (*iii*) the RDF statements link related source objects using predicates from existing or custom vocabularies; (*iv*) duplicate objects, i.e., objects that refer to the same real-world entity, are identified and merged[1]; and (*v*) source objects are linked to objects in external online repositories that refer to the same or related real-world entities. In addition, the output data can be queried efficiently online using the standard SPARQL query language.

In what follows, we present an overview of different components of our framework. Almost all the problems discussed in this paper have been studied in the past to some extent. We present a brief overview of related work in the literature as we explain different components of our framework, although a full discussion of all the related work is beyond the scope of this paper. In terms of the overall framework, our work is related to systems that perform ontology learning and mapping (e.g., The OntoEdit ontology engineering workbench [17], Janus [5], and the work of An et al. [4]), where the goal is semi-automatic or automatic construction of (or mapping to) an ontology from a given set of relational or XML sources and their schemas. In this context, our work can be seen as way of populating several existing ontologies (or creating an ontology) using instances from semistructured sources, in a light-weight approach geared towards creating a high-quality data source following the Linked Data principles. This is similar to what some RDB2RDF systems such as D2R Server [8] perform for relational data (where unlike in semi-structured data the entity types are defined and fixed). Also related to our work is the Haystack system [16] that provides a powerful framework for management of semistructured data in the context of personal information management, without our focus on cleaning, deduplicating, and linking data to the LOD cloud.

2.1 Entity Type Extractor

This component is responsible for extracting entity types and their attributes and relationships from the input data source(s) in a (semi-)automatic manner. The final output of this component is a mapping definition for each entity type. Most semistructured data formats convey an implicit structure that one can leverage to identify the entities and their types in the data. Apart from the implicit structure, there

[1]In certain cases, duplicate objects may be created, but linked with `sameAs` predicates.

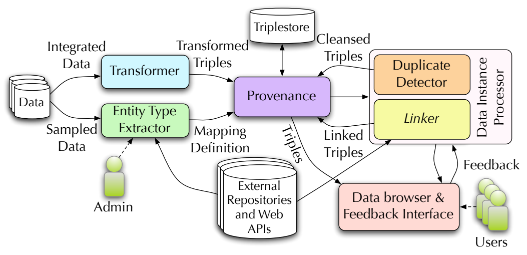


Figure 2: The xCurator Framework

are also several languages designed specifically to describe schemas, such as the widely used DTD, XSD, or RELAX NG schema languages for XML, or the more recent JSON Schema designed for JSON data. In the absence of such schema descriptions, the problem of understanding and extracting a schema for unstructured and semistructured data has been studied extensively in the database literature. Refer to Abiteboul et al. [1] for an overview of early work, and to Bex et al. [6] for a comparison of more recent work.

In our work, unlike the work on schema inference, we are not concerned about deriving a schema that can be used to validate new instances and assist querying the data. Instead, we are interested in discovering the entity types, attributes and relationships, that can be used to generate high-quality Linked Data. Even if explicit schema definitions are given, we need to further process the schema as shown in the example in Section 1. In Section 3, we present the details and challenges of the entity type extraction process, which includes an initial structure extraction phase similar to the above-mentioned inference techniques, followed by refinement steps that enhance the quality of derived entity types.

2.2 Transformer

This component is responsible for transforming the source data into RDF triples based on the mapping definitions generated by the entity type extractor component. The mapping itself is represented and stored in XML with references to source data in XPath, which requires a simple data format transformation layer for sources with non-XML formats. There are several existing tools that transform XML or other semistructured data formats into RDF. Some of these tools are only data format converters, e.g., they convert BibTeX or XML to RDF/XML without properly discovering entity types and their associations. Recently, the XSPARQL language has been proposed as a more concise and intuitive way of defining the mapping between XML and RDF [2]. There are also systems designed to perform transformation of XML data to Linked Data and RDF in domain-specific frameworks that are only capable of handling a fixed XML schema in a domain such as e-government [3] or bibliographic data [9]. One of the main goals of our work is to reduce the burden of having to manually define the mappings or hard-code the transformation procedure.

2.3 Provenance

Since data sources evolve, the entities and their types, relationships and attributes will evolve over time. The Provenance component is responsible for maintaining information about the origins of the data, mapping definitions, and the data life cycle. In our framework, other components can only access the triple store through this component. It is important to note that for all the entities in our framework, we add provenance information as attributes of entities. The challenges involved in handling dynamic semistructured data such as provenance management and the evolution of structure are very similar to those of curated databases (refer to Buneman et al. [11] for an excellent overview of related work in this area).

2.4 Data Instance Processor

After generating triples, this component cleans the data and links the entities to the related entities in both internal and external Web repositories. Basically, this component searches for similar entities in source and external repositories. Then it eliminates duplicates by merging source entities of the same type that refer to the same real-world entity, and links similar entities of different types. The problem of finding duplicate records that refer to the same real-world entity has been extensively studied in the literature [12]. More recently, discovering semantic links in the context of relational databases has also been studied [14]. We use and extend several such existing techniques to enhance the data instance processing component of our framework. Briefly, we take advantage of the entity types extracted in the entity type extraction component and their links to external repositories to enhance the accuracy and efficiency of duplicate detection and link discovery processes. We omit the details in this paper due to space constraints.

2.5 Data Browse and Feedback Interface

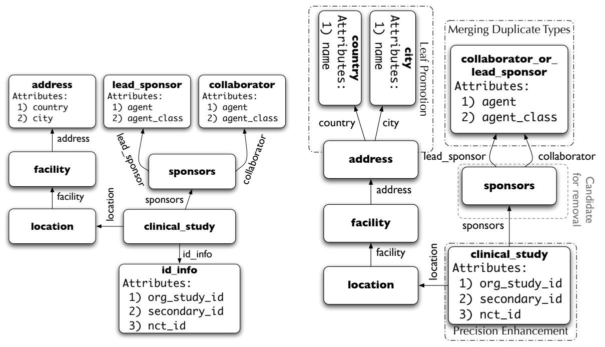
As mentioned earlier, our goal is to publish data following the principles of Linked Data, and allow users to directly query the data by providing a public SPARQL endpoint. In the case of XML input data, we can alternatively (or additionally) embed the RDF data in XML using the standard GRDDL markup format.[2]

In addition to existing RDF and Linked Data browsers that can be used to query and explore the data, we provide a custom data browser mainly with the goal of receiving feedback from users. User feedback is a critical requirement in our system. Entity type identification, duplicate instance detection, and data instance linkage are all problems for which we have only imperfect solutions. No matter how well they are performed, there may still be inaccuracies and inconsistencies in the data that can only be identified and removed using human intelligence. Our goal is to collect and incorporate feedback from users in an efficient and effective way. The data browser interface allows users to log in to our system using their existing OpenID (an open standard for user authentication), provide feedback on the quality of existing data and links, and report missing or erroneous values, duplicate instances, and new external links.

3. ENTITY TYPE EXTRACTION

One of our main goals in xCurator is to provide a generic framework capable of transforming almost any type of semistructured data into rich Linked Data. To achieve genericity, we make no assumption on the availability of a predefined structure (or *schema*) for the input data. Even when such information is available, e.g., through XSD (XML Schema Definition) for XML sources, previous work has shown that for a considerable portion of the XML sources on the Web, either the XML documents are not valid based on the given schema definitions, or the schema definitions

[2]Gleaning Resource Descriptions from Dialects of Languages (GRDDL) `http://www.w3.org/TR/grddl/`



(a) Basic Type Extraction (b) Final Entity Types

Figure 3: Entity type extraction for the example shown in Figure 1

are not valid with respect to W3C standard definitions [7]. This calls for an automatic way of inferring the structure in the data, after which an initial set of entity types can be derived based on the inferred structure.

3.1 Basic Entity Type Extraction

As the first step towards extracting entity types, we can take advantage of the given hierarchy in the data. We do so by first building a *structure graph* from (a sample of) the input data. This graph is similar to the *schema graph* as defined by Abiteboul et al. [1], except that we do not create Schema nodes `Root`, `Any` or type nodes such as `string`. This graph is basically a concise representation of all the possible paths in the input data graph. The structure graph for the example XML data tree in Figure 1 can be derived by simply removing all its leaf nodes (literals). We then consider any non-leaf node in the structure graph as an entity type and the leaf nodes under each non-leaf node as the attributes of its corresponding entity type. Figure 3(a) portrays the result of applying this strategy to extract entity types for the structure shown in Figure 1. As shown in the figure, all the leaf elements (e.g., `<country/>` and `<agent/>`), are mapped to attributes while non-leaf elements (e.g., `<sponsors/>` and `<collaborator/>`) are mapped to entity types. This approach is similar to finding Approximate DataGuides (ADGs) using suffix matching [13] or 1-Representative Objects [18]. Note that this can also be derived from an accurate DTD or XSD for XML data, if available.

In basic entity type extraction, we also find key attributes of the entity types. We do so using a simple map between attribute values and entities in the sample. We omit the details due to lack of space.

3.2 Entity Type Extraction Enhancement

Enhancing the quality of the entity types discovered using the basic approach is quite challenging. Our approach is motivated by the goal of publishing high quality Linked Data. First, we identify and remove duplicate entity types. Next, we identify *accidental* entity types that are likely not useful in linking. Finally, we use external sources to understand if any attributes (leaf nodes) should be promoted to entity types to enable linking.

3.2.1 Duplicate Type Removal

A common problem with the basic entity type identification is that it could result in duplicate types, i.e., different entity types that represent the same real-world entity type, such as `<lead_sponsor/>` and `<collaborator/>` in Figure 3(a). To address this problem, we use a similarity function $f(t_1, t_2)$ for two entity types t_1 and t_2 that returns a similarity value ranging from 0 for no similarity to 1 for highest similarity. We then merge the two entity types t_1 and t_2 if $f(t_1, t_2) \geq \theta$ where θ is a user-defined threshold. This similarity function can be a simple set similarity measure such as the Jaccard coefficient between the sets of attributes and relationships of the two entity types. In Figure 3(a), the Jaccard coefficient between entity types `<lead_sponsor/>` and `<collaborator/>` is 1 and therefore they are merged.

3.2.2 Precision Enhancement

Another problem with the basic type extraction is that there could be non-leaf nodes that do not represent an entity type. Such nodes are usually created for readability or to group entities of the same type. The `<id_info/>` node in Figure 1 is an example of such a node, which is created to group the identifiers of a trial. We detect such nodes based on the cardinality of the relationship of their parent node with them. If the relationship is *injective* (or one-to-one), it implies that removing the node and moving its attributes to its parent is possible, and lossless, meaning that we will not lose any semantic information as a result. For example, the `<id_info/>` type can be removed since there is at most one `<id_info/>` for a trial, but removing the `<collaborator/>` type can result in losing the relationship between the `<agency/>` and `<agency_class/>` attributes of a `<collaborator/>` since there could be several collaborators for each trial. Using this approach, it is also possible to remove the `<sponsors/>` type since each trial has at most one `<sponsors/>` associated with it. However, as our experiments on real data (described in Section 4) show, users may want to keep such type since, for example, the `<sponsors/>` node may actually refer to an entity type that represents "sponsor groups" as opposed to a basic set of sponsors. Our system identifies such nodes based on the frequency of occurrence of each entity for each entity type in the data. In the above example, we keep `<sponsors/>` as an entity type due to the fact that several groups of sponsors frequently occur together.

3.2.3 Recall Enhancement

Another limitation of the basic type extraction method is that many leaf nodes need to be identified as entity types in order to facilitate querying, grouping, deduplication, and linkage. Our approach in xCurator is to identify such types using the power of external knowledge repositories. Our goal is to promote attributes as entity types only if we are able to add additional links from the instances of the entity type to external sources. This general rule has an exception. There are attributes with a very few distinct values (e.g., "Yes"/"No", or "Male"/"Female") for which creating entity types and linking instances may not be appropriate. For such cases, it is reasonable to provide only links between the attribute and the matching entity types, not their instances.

As shown in Algorithm 1, to find linkable attributes, we search a set of knowledge repositories for all distinct values of an attribute. This approach requires using a similarity measure denoted by M to compare the entities. This measure should be symmetric, and preferably translatable to a SPARQL filter.[3] External types which match at least θ_L

[3]These characteristics can significantly improve the performance of recall enhancement in our system.

(linking threshold) fraction of distinct values are considered appropriate links. If we find such links, and the attribute is a key we add the links to the entity type containing the attribute since a key is a representative of the containing entity type. If the attribute is not a key, we promote it to an entity type if it has many distinct values, otherwise we do not promote the attribute.

Algorithm 1: Recall Enhancement Algorithm

```
Input  : $a$ (The attribute), $S$ (Triple repositories), $M$ (The
         similarity measure), $\theta_M$ (The matching threshold),
         $\theta_L$ (The linking threshold), $\theta_V$ (The promotion
         threshold)
/* $V(a)$ is the set of distinct values for attribute $a$, $L$
   is the set of links for an attribute or an entity type,
   $K(t)$, $A(t)$, and $R(t)$ are the set of keys, attributes,
   and relationships for type $t$, respectively.            */
1  $T' \leftarrow \varnothing$;
2  for $v \in V(a)$ do
3      for $r \in S$ do
4          for $(?s\ label\ ?o) \in r$ do
5              if $M(v, ?o) \geq \theta_M$ then
6                  $T'[?t|(?s\ type\ ?t)] \leftarrow T'[?t] + 1$;
7              end
8          end
9      end
10     if $T' = \varnothing$ and this is the first time for spell checking
       then
11         $v \leftarrow spell\_check(v)$;
12         goto line 3;
13     end
14 end
15 $T \leftarrow \{t \in T' | T'[t] \geq \theta_L\}$;
16 if $T \neq \varnothing$ then
17     $t \leftarrow$ The entity type containing $a$;
18     if $a \in K(t)$ or $|V(a)| \leq \theta_V$ then
19         $L(t) \leftarrow T$;
20     else
21         $t_a \leftarrow$ create a new entity type based on $a$;
22         $A(t) \leftarrow A(t) - \{a\}$; $R(t) \leftarrow R(t) \cup t_a$; $L(t_a) \leftarrow T$;
23     end
24 end
```

Another common problem, especially in user-generated data, is the existence of misspellings and alternative representations in attribute values (e.g., misspelled city names in trial locations). To have a better data link quality, if we do not find any type for a value, we check for misspellings and look for alternative representations. This can be done using an internal dictionary depending on the domain. In xCurator, we use an alternative strategy by querying the Google spell checker API for an alternative representation, and then querying the knowledge repositories using the returned value. This has proven to be more effective in our experiments as our source data also comes from the Web.

Algorithm 2: Self Linking Algorithm

```
Input  : $a$ (The attribute), $T$ (The set of entity types),
         $\theta_L$ (The linking threshold)
Output: $L$ (set of entity types which the attribute matches)
/* $f(a, b)$ is a set similarity function such as the Jaccard
   coefficient                                               */
1  for $t \in T$ do
2      for $k \in K(t)$ do
3          if $f(V(a), V(k)) \geq \theta_L$ then
4              $L \leftarrow L \cup \{t\}$;
5          end
6      end
7  end
8  return $L$;
```

A well-known limitation of XML and hierarchical data models is in expressing relationships, a limitation RDF attempts to address. Detecting a comprehensive set of relationships is crucial for generating richly linked data. There are two main styles for expressing relationships in XML and XML-based formats: (*i*) making the data self-contained (e.g., copying/repeating the whole proceedings data for each publication) which increases data redundancy, and (*ii*) implicit referencing where implicit identifiers are used for referencing other elements (e.g., using the ISBN of the proceedings for each publication). Relationships expressed using the first style are detected by the basic type identification while the latter (implicit referencing) cannot be detected without semantics. However, xCurator is capable of extracting those implicitly denoted relationships using Algorithm 2. This algorithm matches an attribute to all the keys of other entity types in order to find implicit references within a document. Note that efficient set-similarity join techniques can be used to make this algorithm scalable to very large data sets.

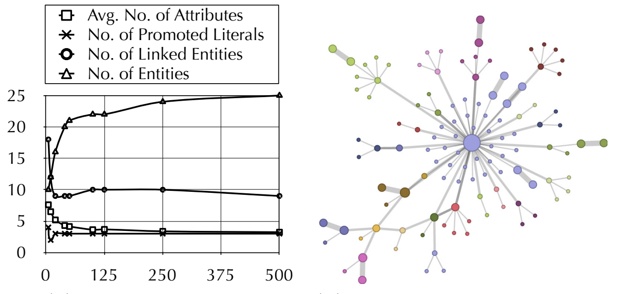
4. EXPERIENCE

In this section, we briefly report our experience using the xCurator framework in transforming several data sets into rich Linked Data.

4.1 Clinical Trials Data

ClinicalTrials.gov is a large repository of clinical trials from all around the world, published by the U.S. National Library of Medicine (NLM). Currently, the data consists of more than 100,000 trials from 174 countries, and is updated regularly. Each trial's data can be retrieved online as a single XML file. In 2008, we manually transformed the data into Linked Data, and have been updating it on average every 6 months. Manually transforming and updating the data has been extremely tedious and time-consuming [15], and one of the main motivations behind the xCurator project. We now have transformed and published the data on the Web using xCurator. The size of the data has grown from around 7 million triples (in the manually transformed data) to more than 24 million triples (using xCurator). Several new entity types have been identified and a few of the entity types have been merged. The automatic linkage has linked instances of entity types that were not identified as *linkable* during the manual transformation. The data has a very complex structure, which makes it particularly suitable for evaluation of our entity type detection techniques.

Figure 4(a) shows the effect of sample size on the quality of the detected entity types. We start with a randomly selected sample of size 5 (five trials) and incrementally add new trials to the sample in order to have comparable samples of different sizes. As depicted in Figure 4(a), xCurator generates almost flat structures (a few entity types having many attributes) when using small sample sizes since many relationships are found to be one-to-one and therefore many entity types are removed. Moreover, a very small sample size introduces many inaccurate external links since the number of distinct values for each entity type is limited. As we increase the sample size, more entities are extracted with each with fewer attributes, as more precise one-to-one relationships are detected. Interestingly, we observe that less than one percent of the data can be used to identify entity types very accurately and that even with only 0.1 percent of the data as a sample, xCurator produces a mapping of much higher quality than our tedious manual mapping. (The accuracy was determined by an expert who has maintained the data for the last three years.) Figure 4(b) portrays the final structure generated by xCurator using the largest sample. The central node is the `clinical_study` entity type, and bold border edges are external links. Similar figures for different sample size and data sets along with more detailed experimental results can be found on our project page [20].



(a) General Statistics (b) LinkedCT Structure Graph

Figure 4: Effects of Sample Size

4.2 Bibliographic Data

DBLP. DBLP data set has been published as Linked Data by multiple sources. DBLP's XML data has a simple, flat structure and therefore is amenable even for manual extraction of entity types. Despite such a straight forward structure, it has two implicit internal links which can be used to evaluate our system: (i) cross references to published collections (e.g., books and proceedings) and (ii) citations to other DBLP-indexed publications. We have performed experiments on identifying entity types which are internally linked together. The results of our experiments show that all publication entity types are properly linked to the entity type for respective collection(s) without manual intervention. Moreover, our system is able to interlink DBLP data with the related topics in Freebase adding value and semantics to even such a heavily curated dataset as DBLP.

BibBase. BibBase started in 2005 at the University of Toronto as a Web service that transforms the BibTeX files of scientists and research groups into good-looking HTML pages with several features such as the ability to use custom style files, group publications based on different attributes, and provide RSS feeds. Recently, we transformed these same BibTeX files into high-quality Linked Data using the capabilities of the xCurator framework. The main application of the xCurator framework for this data source is the data instance processing and the data browse interface. User-generated BibTeX files have various types of quality issues, such as abbreviating author names and using alternative author and conference names. Our Linked Data source currently provides automatic duplicate detection and linkage to external sources. In addition, a group of users are provided with access to the feedback mechanism and are able to report duplicates and external links, and provide feedback on the quality of automatic deduplication and linkage. The results are available online at `http://data.bibbase.org`.

4.3 Mapping Transformation Interface

In addition to the above sources, we provide a light-weight Web interface for users to create custom mappings from (a sample of) their XML data, and transform their data into RDF using the custom mapping. This web interface publicly exposes many features of xCurator to interested users. Users' RDF data will be stored in the xCurator internal triple repository while being accessible by the users.

5. CONCLUSION

In this paper, we presented a modular framework for transforming a semistructured data source into high-quality Linked Data. We described briefly different components of the proposed framework, and the entity type extraction component in more detail. We showed how it can be used for management of several real-world semistructured sources. We are currently investigating several future directions. First, we are planning to further investigate the application of our framework on real-world data sets from various domains. Some applications require additional data format conversion and using more specialized Web repositories for type detection and linkage. We are also in the process of experimentally evaluating the effect of input parameters on the quality and performance of each component in our framework. Another interesting area we are investigating is effective generation of representative samples from a given data source that result in more efficient mapping generation and could also assist users of the system in updating the given mapping. The results of our evaluation and the output data sources will be made available online [20].

6. REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[2] W. Akhtar, J. Kopecký, T. Krennwallner, and A. Polleres. XSPARQL: Traveling between the XML and RDF Worlds - and Avoiding the XSLT Pilgrimage. In *ESWC 2008*.

[3] F. Amato, A. Mazzeo, A. Penta, and A. Picariello. Building RDF Ontologies from Semi-Structured Legal Documents. In *CISIS 2008*.

[4] Yuan An, Alex Borgida, and John Mylopoulos. Discovering and Maintaining Semantic Mappings between XML Schemas and Ontologies. *J. of Comp. Sci. & Eng.*, 2(1):44–73, 2008.

[5] Ivan Bedini. *Deriving Ontologies Automatically from XML Schemas Applied to the B2B Domain*. PhD thesis, University of Versailles, France, 2010.

[6] G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning Deterministic Regular Expressions for the Inference of Schemas from XML Data. In *WWW 2008*.

[7] G. J. Bex, F. Neven, and J. V. den Bussche. DTDs versus XML Schema: A Practical Study. In *WebDB 2004*.

[8] C. Bizer and R. Cyganiak. D2R Server - Publishing Relational Databases on the Semantic Web. Poster at *ISWC 2006*.

[9] C. Bizer, R. Cyganiak, and T. Gauss. The RDF Book Mashup: From Web APIs to a Web of Data. In *SFSW 2007*.

[10] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data: Principles and State of the Art. In *WWW 2008*.

[11] P. Buneman, J. Cheney, W. C. Tan, and S. Vansummeren. Curated Databases. In *PODS 2008*.

[12] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate Record Detection: A Survey. *IEEE TKDE*, 19(1):1–16, 2007.

[13] R. Goldman, J. McHugh, and J. Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. In *WebDB 1999*.

[14] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R. J. Miller, and M. Wang. A Framework for Semantic Link Discovery over Relational Data. In *CIKM 2009*.

[15] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R. J. Miller, and M. Wang. LinkedCT: A Linked Data Space for Clinical Trials. *CoRR*, abs/0908.0567, 2009.

[16] D. R. Karger, K. Bakshi, D. Huynh, D. Quan, and V. Sinha. Haystack: A General-Purpose Information Management Tool for End Users Based on Semistructured Data. In *CIDR 2005*.

[17] A. Maedche and S. Staab. Ontology Learning for the Semantic Web. *IEEE Intell. Systems*, 16(2):72–79, 2001.

[18] S. Nestorov, J. D. Ullman, J. L. Wiener, and S. S. Chawathe. Representative Objects: Concise Representations of Semistructured, Hierarchial Data. In *ICDE 1997*.

[19] S. S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. Thibodeau Jr, S. Auer, J. Sequeda, and A. Ezzat. A Survey of Current Approaches for Mapping of Relational Databases to RDF. Technical report, W3C RDB2RDF incubator group, 2009.

[20] The xCurator Project Homepage. `http://dblab.cs.toronto.edu/project/xcurator`.