Data.dcs: Converting Legacy Data into Linked Data*

Matthew Rowe
OAK Group
Department of Computer Science
University of Sheffield
Regent Court, 211 Portobello Street
S1 4DP Sheffield, United Kingdom
m.rowe@dcs.shef.ac.uk

ABSTRACT

Data.dcs is a project intended to produce Linked Data describing the University of Sheffield's Department of Computer Science. At present the department's web site contains important legacy data describing *people*, *publications* and *research groups*. This data is distributed and is provided in heterogeneous formats (e.g. HTML documents, RSS feeds), making it hard for machines to make sense of such data and query it. This paper presents an approach to convert such legacy data from its current form into a machine-readable representation which is linked into the Web of Linked Data. The approach describes the *triplification* of legacy data, coreference resolution and interlinking with external linked datasets.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: General

General Terms

Linked Data

Keywords

Linked Data, Triplification, Coreference Resolution

1. INTRODUCTION

Recent work has addressed the issue of producing linked data from data sources conforming to well-stuctured relational databases [2]. In such cases data already follows a logical schema making the creation of linked data a case of schema mapping and data transformation. The majority of the Web however does not conform to such a rigid representation, instead the heterogeneous structures and formats which it exhibits makes it hard for machines to parse and interpret such data. This therefore makes the process of producing linked data limited.

In this paper we use the case of the University of Sheffield's Department of Computer Science (DCS). The DCS web site contains information about people - such as their name, email address, web address and location, research groups and publications. The department provides a publication database located separately from the main site on which DCS members manually upload their papers. Each member of the department is responsible for their own personal web page, this has lead to the formatting and presentation of legacy data to vary greatly between pages, where some pages contain RDFa and others are plain HTML documents with the bare minimum of markup. This impacts greatly on the usability of the site in general and the slow process by which information can be acquired. For instance finding all the publications which two or more research groups have worked on in the past year would take a large amount of filtering and data processing. Furthermore the publication database is rarely updated to reflect publications by the department and its members.

This use case presents a clear motivation for generating a richer representation of legacy data describing the DCS. We define *legacy data* as data which is present in proprietary formats and which describes important information about the department - i.e. publications. Leveraging legacy data from HTML documents which make up the DCS web site and converting this data into a machine-readable form using formal semantics would link together related information. It would link people with their publications, research groups with their members and allow co-authors of research papers to be found. Furthermore by linking the dataset into the Web of Linked Data would allow additional information to be inferred such as the conferences which members of the DCS have attended and provide up-to-date publications listings - thereby avoiding the current slow update process by linking to popular bibliographic databases such as DBLP.

In this paper we document our current efforts to convert this legacy data to linked data. We present our approach to pursue this goal which is comprised of three stages: first we perform triplification of legacy data found within the DCS - by extracting person information from HTML documents and publication information from the current bibliography system. Second we perform coreference resolution and interlinking of the produced triples - thereby linking people with

*The research leading to these results has received funding from the EU project WeKnowIt10 (ICT-215453).

LDOW2010, April 27, 2010, Raleigh, USA.

their publications and fusing data within separate HTML documents together. Third we connect our produced dataset to distributed linked datasets in order to provide additional information to agents and humans browsing the dataset.

We have structured the paper as follows: section 2 describes related work in the field of producing linked data from legacy data and discusses similar efforts to our problem setting explored within the information extraction community. Section 3 presents a brief overview of our approach and the pipeline of the architecture which is employed. Section 4 describes the triplication process which generates triples from legacy data within HTML documents and the publication database. Section 5 presents the SPARQL rules we employed to discover coreferring entities. Section 6 describes our preliminary method for weaving our dataset into the linked data cloud. Section 7 finishes the paper with the conclusions which we have learnt from this work and our plans for future work.

2. RELATED WORK

Recent efforts to construct linked data from legacy data include *Sparqplug* [4] where linked data models are constructed based on Document Object Model (DOM) structures of HTML documents. The DOM is parsed into an RDF model which then permits SPARQL [11] queries to be processed over the model and relevant information returned. Although this work is novel in its approach to semantifying web documents, the approach is limited by its lack of rich metadata descriptions attributed to elements within the DOM. Existing work by [2] presents an approach to expose linked data from relational databases by creating lightweight mapping vocabularies. The effect is such that data which previously corresponded to a bespoke schema is provided as RDF according to common ontological concepts. Metadata generation - so called *triplification* - is discussed extensively in [10] in order to generate metadata describing conferences, their proceedings, attendees and organisations participating. Due to the wide variation in the provided data formats - i.e. excel spreadsheets, table documents - metadata was generated by hand. Despite this such work provides a blue print for generating metadata by describing the process in detail and the challenges faced.

The challenges faced when converting legacy data devoid of metadata and semantic markup into a machine-processable form involves exposing such legacy data and then constructing metadata models describing the data. In the case of the DCS web site our goal is to generate metadata describing members of the department, therefore we must extract this legacy data to enable the relevant metadata descriptions to be built. Work within the field of information extraction provides similar scenarios to the problems which we face, For instance extraction of person information from within HTML documents has been addressed in [14] by segmenting HTML documents into components based on the Document DOM of the web pages. Person information is then extracted using induced wrappers from labelled personal pages. [15] uses manually created name patterns to match person names within a web page and then, using a context window surrounding the match, extract contextually relevant information surrounding the name. The DOM of HTML documents is utilised in work by [17] to provide clues to regions within the documents from which person information should be extracted. Once regions of extraction have been identified then extraction patterns are used to extract relevant information based on its proximity in the document. An effort to extract personal information (name, email, homepage, telephone number) from within web pages has been presented in [3] using a system called "Armadillo". A lexicon of seed person names is compiled from several repositories which are then used to guide the information extraction process. Heuristics are used to extract person information surrounding a name which appears within a given web page.

Work by [18] has explored the application of Hidden Markov Models to extract medical citations from a citation repository by inputting a sequence of tokens and then outputting the relevant labels for those tokens based on the HMM's predicted states: Title, Author, Affiliation, Abstract and Reference. Prior to applying the HMMs, windows within HTML documents are derived known as *component zones*, or *context windows*, these zones within the HTML document are considered for analysis in order to extract information from. Similar work by [7] has applied HMMs to the task of extracting citation information. Work within the field of attribute extraction has placed emphasis on the need to extract information describing a given person from within web pages. For instance [9] uses extraction patterns (i.e. regular expressions) defined for different person attributes to match content within HTML documents. An approach by [16] to extract person attributes from HTML documents first identifies a list of candidate attributes within a given web page using hand crafted regular expressions - these are related to different individuals. All HTML markup is then filtered out leaving the textual content of the documents. Attributes which appear closest to a given person name are then assigned to that name.

3. CONVERTING LEGACY DATA INTO LINKED DATA

In order to convert legacy data into linked data we have implemented a pipeline approach. Figure 1 shows the overview of this approach which is divided into three stages:

- *Triplification*: the approach begins by taking as input an RSS feed describing the publications by DCS members and the DCS web site. Context windows are identified within the RSS feed - where each context window contains information about a single publication - and in the HTML documents - where each context window contains information about a single person. Information is extracted from these context windows and is then converted into triples, describing instances of people and publications within the department.

- *Coreference Resolution*: SPARQL queries are processed over the entire graph to discover coreferring entities: e.g. the same people appearing in different web pages.

- *Linking to the Web of Linked Data*: the Web of Linked Data Cloud is queried for coferring entities and related information resources, and links are created from the produced dataset.

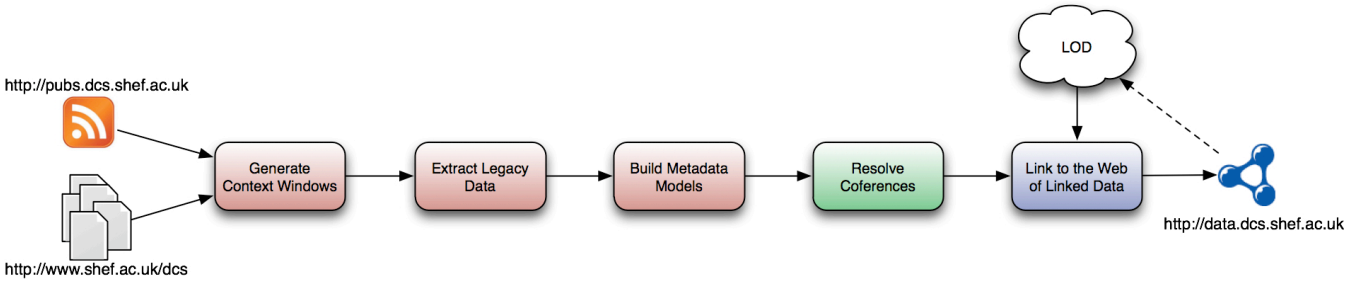

Figure 1: Three staged approach to convert legacy data to linked data

Each of the stages of the approach contains various steps and processes which are essential to the production of a linked dataset. We will now present each of these stages in greater detail, beginning with the *triplication* of legacy data.

4. TRIPLIFICATION OF LEGACY DATA

The DCS web site contains listings of members of the department: staff, researchers and students, and their associated information (name, email address, web address) provided within HTML documents. Such documents lack metadata descriptions which limits the applicability of automated processes to parse and interpret the data. Therefore we require some method to leverage legacy data which can then be converted into triples to allow machine-processing, for instance by associating a person with his/her name, email address, etc. For publications we are confronted with a slightly different problem. We are provided with an RSS feed[1] containing the publications within the department, this feed should be well structured with declarative elements for each attribute of a publication (i.e. title, authors, year, etc). Instead we are returned the following:

```
<title>Interlinking Distributed Social Graphs</title>
<link>http://publications.dcs.shef.ac.uk/show.php?record=4161</link>
<description>
  <![CDATA[Rowe, M. (2009). Interlinking Distributed Social Graphs.
  In <i>Proceedings of Linked Data on the Web Workshop, WWW 2009,
  Madrid, Spain. (2009)</i>. Madrid, Madrid, Spain.<br>
  <br>Edited by Sarah Duffy on Tue, 08 Dec 2009 09:31:30 +0000.]]>
</description>
<pubDate>Mon, 07 Dec 2009 17:03:27 +0000</pubDate>
<author>Sarah Duffy &lt;s.duffy@dcs.shef.ac.uk&gt;</author>
```

In the above XML the `<title>` element contains the title of the paper, however other paper attributes are not placed within suitable elements - i.e. using `<author>` element for the author of the paper. Instead all the data which describes the paper is stored within the `<descrption>` element. A technique is required which is able to extract information from the `<description>` element which corresponds to the relevant attributes of the paper, for instance by extracting "Interlinking Distributed Social Graphs" for the title attribute.

Unlike publications however, extracting person information from HTML documents requires the derivation of a *context windows* which contain person attributes - this akin to being provided with the content within the above `<description>` element. We must identify such context windows within a HTML document to enable the correct information to be extracted. To address this problem we rely on the markup used within HTML documents to segment disjoint content. For instance in many web pages layout elements such as `<div>` elements are used to contain information about a single entity. Another `<div>` element is then used to contain information about another entity. Using such elements provides the necessary means through which context windows can be identified - through the use of layout elements within a DOM - and information extraction techniques can be applied to leverage the legacy data. We now explain how we generate context windows from HTML documents.

4.1 Generating Context Windows

To derive a set of context windows from a given HTML document, we first tidy the HTML document into a parseable form using Apache Maven's HTML Parser[2]. HTML is often messy and contains poorly structured markup where HTML tags are opened and not closed. This reduces its ability to be parsed where such techniques require a well-formed DOM. Once tidied the DOM is used as input to Algorithm 1 as follows: first a list of name patterns is loaded and applied to the DOM model, for each pattern the list of DOM elements which that pattern matches are collected (line 5). The pattern list contains a set of regular expressions designed to detect the appearance of a person name within a given body of text (e.g. `<first-name> <capitalised-word>+`). Each of the collected DOM elements is then verified as having not been been processed before (line 6) - as different name patterns may match the same person name at the same position in the document. The trigger string is extracted from the element (line 8) noting the person's name that was matched using the name pattern. The parent node type of the DOM element (e) is then assessed to see if it is a hyperlink: it is common for a person name to appear within a HTML document as a hyperlinked element. If it is hyperlinked then the grandparent of the element is considered as a possible area from which the context window can be gathered. However should the parent node of the element (e) not be hyperlinked (line 12) then the parent is then passed onto the *domManip* function for assessment together with the trigger string.

Algorithm 2 (*domManip*) takes the trigger node and a node from within the DOM and manipulates the DOM structure to derive a suitable DOM element from which the context window should be derived. First the node type is checked

[1]http://pubs.dcs.shef.ac.uk

[2]http://htmlparser.sourceforge.net/

Algorithm 1 cwFind(*dom*) : Given the DOM of a HTML document, returns a set of context windows

Input: *dom*
Output: Set of context windows C

```
1:  N =person name patterns
2:  C = ∅
3:  visited = ∅
4:  for each n ∈ N do
5:      E = getElements(dom, n)
6:      for each e ∈ E do
7:          if e.startIndex ∉ visited then
8:              trig = extract(e, n)
9:              if e.parent.type ==< a > then
10:                 c = domManip(e.parent.content, e.parent.parent)
11:                 C = C ∪ c
12:             else
13:                 c = domManip(trig, e.parent)
14:                 C = C ∪ c
15:             end if
16:             visited = visited ∪ e.startIndex
17:         end if
18:     end for
19: end for
20: return C
```

Algorithm 2 domManip(*trig*,*node*) : Given a trigger string and the node which contains the trigger, derives the suitable DOM element to extract the window from

Input: *trig* and *node*
Output: *window*

```
1: if node.type ==< td > then
2:     window = extractWin(trig, node.parent.content.substring(trig))
3: else if node.type ==< style > then
4:     domManip(trig, node.parent)
5: else
6:     window = extractWin(trig, node.content.substring(trig))
7: end if
```

to see if it is a <*td*> element - denoting that the trigger appeared within a table in the HTML document (line 1). If this is the case then the trigger string is passed to *extractWin* together within the parent of the <*td*> element: the <*tr*> element which <*td*> is a child of. If the node is a style element (<*b*>,<*h1*>,<*span*>, etc) (line 3) then *domManip* is recursively called using the trigger and the parent node. Such elements control the presentation and styling of a HTML document but do not control or segment the layout like <*div*>, <*li*> or <*td*> elements do. If neither of the above cases are true, and the node is a layout element via the process of elimation (line 5) then the content of the node and the trigger string is passed on to the window extractor. It is worth noting that the content of the nodes which is passed onto *extractWin* contains HTML markup along with textual content. Unlike exisiting work within the attribute extraction state of the art, markup is maintained as it provides clues which can aid the process of information extraction (i.e. HTML tags acting as delimiters between person attributes).

Algorithm 3 (*extractWin*) takes the trigger string (i.e. the person name) and HTML content string and derives the context window. First a mapping set is initialised and the list of person name patterns is loaded (lines 1-2). The preamble of the HTML content string contains the trigger string, therefore this is removed to enable the name patterns to match the remaining content. Each pattern is applied to the content string (line 4), if a match occurs (line 5) then the name pattern is added to the mapping along with the index within

Algorithm 3 extractWin(*trig*,*content*) : Given a trigger string and a DOM element's content, extracts the window from the trigger onwards

Input: *trig* and *content*
Output: *window*

```
1:  maps = ∅
2:  N = person name patterns
3:  remove(content, trig)
4:  for each n ∈ N do
5:      if match(content, n) then
6:          maps = maps∪<n.startMatchIndex, n>
7:      end if
8:  end for
9:  if |maps| > 0 then
10:     order(maps)
11:     <i, n> = maps_1
12:     return trig + content.substring(0, i)
13: else
14:     return content
15: end if
```

the content string where the pattern match starts. Once all patterns have been applied to the content string, the mappings, if there are any, are ordered by their start matching points within the content string. The first mapping is then chosen from the ordered set of mappings, given that this provides the nearest point to the start of the content string where a person name appears. The content string is then removed of the content following the earlier match (line 12), the trigger string is appended back on to the start and this is returned as the context window. Should no mappings be found (line 13) then the content string is returned as the context window.

The derived context window from *extractWin* feeds back to *cwFind* and populates the set of context window set for the given HTML document. These algorithms for context window derivation provide a conservative strategy to identify areas of a HTML document from which person information can be extracted. It is conservative in the sense that it does not look above certain DOM element types (i.e. <*div*>) instead it relies on the logical segmentation of the document to provide the necessary features which can be utilised to identify context windows. Applying this approach to a HTML document containing the following markup would by triggered by the person name *Matthew Rowe*, the algorithms would traverse two nodes up from the element containing the trigger string until a <*div*> element is found. The context window is then returned as the substring of the content within the <*div*> element from the trigger string - the person name *Matthew Rowe* onwards - until the end of the <*div*> element's content, returning the following:

```
<div>
  Matthew Rowe</h4>
  <img src="../images/people/matt.jpg"/>
  <p class="position">Ph.D. Student</p>
  <ul>
    <li>
      <a href="http://www.dcs.shef.ac.uk/ mrowe/">
        http://www.dcs.shef.ac.uk/ mrowe/
      </a>
    </li>
    <li>
      <a href="mailto:m.rowe@dcs.shef.ac.uk">
        M.Rowe@dcs.shef.ac.uk
      </a>
    </li>
  </ul>
```

```
  <p>Researching Identity Disambiguation and Web 2.0</p>
</div>
```

4.2 Extracting Legacy Data using Hidden Markov Models

Given a set of context windows derived from a HTML document person information must now be extracted from the windows. Person information consists of four attributes: *name*, *email*, *web page* and *location*. The appearance and order in which these attributes appear in the context window can vary (e.g. (*name*,*email*,*www*) or (*email*,*name*,*location*). Context windows for publications are also provided using the content from all of the `<description>` elements within the RSS publication feed. Publication information also consists of four attributes: *title*, *author*, *year* and *book*. We use the *bookTitle* attribute to define where the publication appears, this could be a thesis - in which case it would be the university - or a journal paper - in which case it would be the name of the journal publisher.

In order to extract both person and publication information from their relevant context windows Hidden Markov Models (HMMs) are used. HMMs provide a suitable solution to this problem setting by taking as input a sequence of observations (e.g. tokens within a context window) and outputting the most likely sequence of states where each state corresponds to a piece of information to be extracted. HMMs use Markov chains to work out the likelihood of moving from one state to another ($s_i \rightarrow s_j$) and outputting symbol (σ when in state s_j). A HMM is described as hidden in that it is given a known sequence observations with hidden states, it must therefore label these hidden states which correspond to person or publication attributes which are to be extracted. A HMM consists of a set of States; $S = \{s_1, s_2, ..., s_m\}$, a vocabulary of symbols; $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_n\}$, a transition probability matrix; A (where $a_{ij} = P(s_j|s_i)$), an omission probability matrix; B (where $b_{i\sigma} = P(\sigma|s_i)$) and a start probability vector (where π where $\pi_i = P(s_i|s_{start})$). These parameters must be built, or estimated, from known information, essentially training the HMM from previous context windows to allow information to be extracted from future context windows.

4.2.1 HMM States

The topology of the HMM defines what states are to be used and how those states are connected together. States within the HMM fall into two categories: *major* and *minor* states. For person information extraction there are 4 *major* states which constitute the four person attributes. 13 *minor* states are defined in order to provide clues to the HMM and enhance the process of deriving the state sequence. Of those 13 *minor* states there are 2 *pre-major* states (*pre email* and *pre www*), 10 *separator* states (e.g. *between email and name*) and 1 *after* state which contains the symbols omitted at the end of the window. Omissions made within the minor states offer clues as to the order of the state sequence and what information is to follow. For instance using the omission of the token $<a$ would indicate that a proceeding field might contain an email address or a web address. There is also a single *start* state in which the HMM begins. The start probability vector (π) contains the transition probabilities of moving from this state to another given state. Similar to the person topology, the topology of the HMM for publication information extraction also contains 4 *major* states corresponding to the four publication attributes. 3 *Minor* states are used to separate the *major* states and a single *after* state is included. Figure 2 shows the topology of the HMM used for publication information extraction.

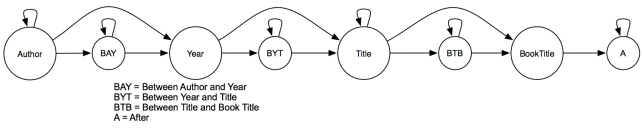

Figure 2: Topology of HMM for publication information extraction

4.2.2 Parameter Estimations

Once the states have been decided for the information attributes the remaining parameters of the HMM must be estimated. We must train the HMM to detect what transitions are more likely than others and to calculate the probability of omitting a given symbol whilst in a given state. The transition probability matrix A is built from labelled training data by counting the number of times a given state s_i transits to state s_j. This count is then normalised by the total number of transitions from state s_i. Formally A is populated as follows:

$$a_{ij} = \frac{c(s_j|s_i)}{\sum_{s \in S} c(s|s_i)}$$

Similar to A, the omission probability matrix B is built from labelled training data. Counts are made of how many times a given symbol is observed in a given state, this count is then normalised by the total number of symbols omitted in that state. B is therefore defined as follows:

$$b_{i\sigma_n} = \frac{\sigma_n|c(s_i)}{\sum_{\sigma \in \Sigma} c(\sigma|s_i)}$$

The start probability vector is built from the training data by counting how many times a given state is started in. This is then normalised by the total number of start states observed:

$$\pi_i = \frac{c(s_i|s_{start})}{\sum_{s \in S} c(s|s_{start})}$$

4.2.3 Smoothing

When estimating the parameters of the HMM from labelled training data it is likely that certain state-to-state transitions, or omissions whilst in a given state are not observed. The trained HMM, when applied to test data, may find previously unknown paths, or symbols omitted in states which have previously not been witnessed. The model must be able to deal with such possibilities by smoothing the transition and omission probabilities to cope with unseen observations and transitions within the training data. One such smoothing technique is known as *Naive Smoothing* [7] . Naive Smoothing functions by setting zero probabilities in A and B to a very low constant of 10^{-7} . A and B are estimated using normalised values such that $\sum_{j=1}^{|A|} a_{ij} = 1$ and

$\sum_{k=1}^{|B|} b_{i\sigma_k} = 1$, therefore when smoothing the zero probabilities in both A and B the non-zero probabilities must also be adjusted to ensure that that the distributions hold. Therefore all non-zero probabilities have the value of $\frac{n}{m}10^{-7}$ subtracted from the current value where n is number of zero-probability events and m is the number of non-zero probability events.

Another smoothing method using *Additive Smoothing* (*Laplace Smoothing*) described in [6] increments all zero counts when building A and B by 1. This ensures that the respective transitions and omissions are then assigned a low probability which is non-zero. It is worth noting that the use of smoothing is only applicable if the training data does not sufficiently cover possible transitions which are likely to appear and observations which are present in test data.

4.3 Vocabulary Dimension Reduction

One of the parameters of a HMM is its vocabulary of symbols - where the term symbol is used to refer to a given observation i.e. word, token, etc. This vocabulary contains all the possible observations or omissions which might be found within an input sequence. In [18] the vocabulary is compiled from a large corpus of words which make up all the possible symbols that could be observed. This works well where the dictionary is a finite size, however in the case of HTML markup leads to new combinations. To solve this problem we control the dimension of the vocabulary that only a fixed number of symbols are used. Dimension control is performed using transformation functions as follows: a given input - i.e. the context window - is tokenized, each token is then transformed into its respective symbols using transformation functions. The transformed input sequence of symbols is then used to derive the correct state sequence. The vocabulary contains 16 symbols, where each symbol has a transformation function which matches the token to a given symbol. For instance the symbol *First_Name (FN)* is used to identify a person's name. Symbols are also used for different HTML tags such as an opening tag (e.g. $<a$). Web data is noisy and contains a large amount of variation in content form. Controlling the vocabulary of symbols allows previously unseen tokens to be handled appropriately.

4.4 Deriving Transition Paths

Given a tokenized context window which has been converted into symbols, the Viterbi algorithm [5] is then used to calculate the most probable state path through the window. This path is found using A and B: given the sequence of observations the path is returned composed of the maximum likelihood estimates of moving from one state to another and then omitting a given symbol. The Viterbi algorithm uses the learnt HMM, and its estimated parameters, as background knowledge of known transitions and omissions and assesses the input sequence to find clues as to the order of states. This allows consistencies in the layout and presentation of person information to be utilised to extract information for future tasks. For instance, it is common for a person to hyperlink their name with their web address, learning such patterns allows for future similar information extraction tasks to be recognised and the correct information extracted.

4.5 Evaluation

The success of our *triplification* technique depends on its ability to extract the maximum amount of person and publication information whilst ensuring that the extracted legacy data is accurate and contains no errors - as this could be detrimental to the linking of this data into the Web of Linked Data. Therefore we evaluate our triplification approach using the evaluation measures precision and recall defined as $precision = |A \cap B|/|B|$ and $recall = |A \cap B|/|A|$ where A denotes the set of relevant tokens, and B denotes the set of retrieved tokens. Precision measures the proportion of tokens which were labelled correctly. Recall measures the proportion of correct tokens which were found. These measures gauge the accuracy of the labels and the ability of the technique to find person information within the HTML document - as lower levels of recall indicate that person information is missed. Evaluation is therefore performed on a per token basis for person information extraction within a given HTML document and per token basis within the publication feed for publication information extraction by assessing the accuracy of the HMM in labelling tokens with their respective major state labels and the ability of the technique to detect context windows. F-measure (referred to in the results as F1) provides the harmonic mean of precision and recall as follows:

$$f - measure = \frac{2 \times precison \times recall}{precision + recall}$$

The evaluation dataset was compiled by crawling the Department of Computer Science web site[3] - in a similar vein to work by [3]. All internal pages within the web site were collected, totaling 12,000 HTML documents, of this collection 3,500 documents were found to contain person information. Context windows were derived for each of these documents and 200 randomly selected context windows were used as training data for the HMM. Each window is already tokenized, however for training each token is labeled with the state in which it appears. The test data was also compiled by randomly selecting 40 URLs from the dataset and their respective context windows, therefore totalling 203 context windows. A gold standard was then created for these windows by manually labelling the tokens within their respective states. Each URL was also manually analysed to find context windows which were missed by the context window derivation algorithms, these were then added to the gold standard. We performed the same setup for publications by generating 200 tokenised context windows - using content from `<description>` elements in the publication RSS feed - and labelling each of the tokens in each window with its respective states for training and choosing another 200 windows randomly for testing.

4.5.1 Results

As the results from Table 1 and Table 2 show Naive Smoothing achieves, on average, higher f-measure levels with respect to the alternative smoothing method. Additive Smoothing yields poorer scores, particularly for labelling web addresses and locations. Both smoothing techniques perform poorly in terms of recall when extracting location information. In terms of publication information extraction the results are

[3] http://www.dcs.shef.ac.uk

Table 1: Accuracy levels of extracting person information using Hidden Markov Models with different smoothing methods

	Naive Smoothing			Additive Smoothing		
Attribute	P	R	F1	P	R	F1
Name	0.903	0.875	0.889	0.928	0.703	0.8
Email	1	0.867	0.928	0.578	0.688	0.628
WWW	0.849	0.833	0.841	0.714	0.714	0.714
Location	0.888	0.444	0.592	0.421	0.211	0.281
Average	**0.910**	**0.754**	**0.825**	**0.66**	**0.579**	**0.616**

Table 2: Accuracy levels of extracting publication information using Hidden Markov Models with different smoothing methods

	Naive Smoothing			Additive Smoothing		
Attribute	P	R	F1	P	R	F1
Title	0.941	0.698	0.801	0.901	0.589	0.712
Year	1	0.716	0.835	1.000	0.678	0.808
Author	0.952	0.717	0.818	0.934	0.687	0.792
Book Title	0.982	0.652	0.783	0.956	0.500	0.657
Average	**0.969**	**0.696**	**0.810**	**0.948**	**0.614**	**0.745**

similar to the performance when applying HMMs for person information extraction. Naive smoothing yields higher f-measure scores overall and almost perfect precision - indicating that the extracted information rarely contains mistakes. However particularly for the paper title and the book title several tokens are missed leading to incomplete titles. This is something which must be addressed in future work as the error will scale up to become detrimental to data quality.

4.6 Building RDF Models from Legacy Data

Using HMMs together with Naive Smoothing we build an RDF dataset describing all instances of people and publications within the department. This dataset provides the *source* dataset from which we build our *linked dataset* for deployment. We apply the above techniques to the entire dataset collected from the DCS web site in order to build metadata models describing person information found within each web document. We also apply the technique to build RDF models describing publications within the department. In each case we use temporary URIs to provide unique RDF instances constructed from the extracted legacy data. We use a namespace to identify the RDF instance as denoting a person `http://data.dcs.shef.ac.uk/person/` and append an incremented integer to form a new URI for a given person. For each person found within a given HTML document we create instances of *foaf:Person* and assign their name to the instance using *foaf:name*, hashed emailed address using *foaf:sha1_sum* and homepage using *foaf:homepage*. We associate the person instance to the web page within the department's web site on which the instance appeared using *foaf:topic*. An example instance of *foaf:Person* is as follows (using Notation 3 syntax).

```
<http://data.dcs.shef.ac.uk/person/12025>
  rdf:type foaf:Person ;
  foaf:name "Matthew Rowe" .
<http://www.dcs.shef.ac.uk/~mrowe/publications.html>
  foaf:topic <http://data.dcs.shef.ac.uk/person/12025>
```

For publications we model extracted information using the Bibtex ontology[4] by creating an instance of *bib:Entry* for each publication instance. We use a temporary URI for each publication instance by taking the publication namespace `http://data.dcs.shef.ac.uk/paper/` and appending an incremented integer for each each new publication. We then assign the relevant attributes to the instance using concepts from the Bibtex ontology. For the title we use *bib:title*, for the year we use *bib:hasYear* and for the book title we use *bib:hasBookTitle*. For each paper author we create a blank node typed as an instance of *foaf:Person* and assign the author name to the instance using *foaf:name* and associate this instance with the publication instance using *foaf:maker*. Referring back to the example from the beginning of this section, the RSS feed provided by the publication base contained publication information - containing all four attributes - within a single `<description>` element. This legacy data, once extracted and converted into triples, is provided as follows (again using Notation 3 syntax):

```
<http://data.dcs.shef.ac.uk/paper/239>
  rdf:type bib:Entry ;
  bib:title "Interlinking Distributed Social Graphs." ;
  bib:hasYear "2009" ;
  bib:hasBookTitle "Proceedings of Linked Data on the
    Web Workshop, WWW , Madrid, Spain." ;
  foaf:maker _:a1 .
_:a1
  foaf:name "Matthew Rowe"
```

5. COREFERENCE RESOLUTION

Following conversion of the DCS web site and publication database we are provided with an RDF dataset containing 17896 *foaf:Person* instances and 1088 *bib:Entry* instances. Using this dataset we must discover coreferring instances such as equivalent people appearing in separate web pages and identify publications which people have published. This stage in the approach starts the process of compiling the *linked dataset* which will be deployed for consumption. Therefore we perform *coreference resolution* to identify equivalent instances in the dataset and fuse data together - this will provide rich instance descriptions when a resource is looked up in our *linked dataset*.

5.1 Building Research Groups

Our produced *linked dataset* is intended to contain information about research groups and their members and publications. Therefore we generate an instance of *foaf:Group* for each research group and assign the group a minted URI using the group namespace `http://data.dcs.shef.ac.uk/group` and appending an abbreviation of the group name (e.g. `nlp` for the Natural Language Processing Group). We then assign a name to the group using *foaf:name* and the URL of the group web page using *foaf:workplaceHomepage*. This produces the following:

```
<http://data.dcs.shef.ac.uk/group/oak>
  rdf:type  foaf:Group  ;
  foaf:name  "Organisations, Information and Knowledge
    Group" ;
  foaf:workplaceHomepage  <http://oak.dcs.shef.ac.uk>
```

[4]http://zeitkunst.org/bibtex/0.1/bibtex.owl#

Once we have constructed all of the group instances we then query our *source* dataset for all the people who appear on each of the group personnel pages. This provides us with the members of the DCS whose information is going to the compiles and deployed as linked data. This step in the approach acts as seeding the forthcoming coreference resolution processes by compiling a set of members. It worth noting however that in doing so we are only considering a subset of the entire collection of *foaf:Person* instances. We plan to analyse this data in future work, however for now we are concerned with producing linked data describing the DCS.

5.2 Person Disambiguation

We are provided with a set of people who are members of the DCS, who either work or study there. We perform person disambiguation to identify other instances of *foaf:Person* in separate web documents which are in fact the same people as the DCS members. Our first person disambiguation method uses *Instance Smushing* [13] to discover equivalent instances. This technique works by matching resources associated with disparate RDF instances where the resources are associated with the instances using properties which are defined as *owl:inverseFunctionalProperty*. An example of instance smushing is the identification of equivalent person instances using the email address of the person. In essence *Instance Smushing* uses the declarative characteristics of such properties to detect coreference. We smush instances of *foaf:Person* which appear on research group personnel pages using the following SPARQL rule for *foaf:homepage* property:

```
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
CONSTRUCT {
  ?x owl:sameAs ?y .
  ?x foaf:page ?p
}
WHERE {
  <http://oak.dcs.shef.ac.uk/people> foaf:topic ?x .
  ?x foaf:homepage ?h .
  ?p foaf:topic ?y
  ?y foaf:homepage ?h
  FILTER (<http://oak.dcs.shef.ac.uk/people> != ?p)
}
```

The triple within the `CONSTRUCT` clause infers an *owl:sameAs* relation between a member of the oak group and another person instance on a separate web page, and infers that the page cites the group member - expressed using *foaf:page*.

Our second person disambiguation technique employs person co-occurence to identify coreferring instances of *foaf:Person*. We assume that if a group member appears on a web page with a coworker then that page will refer to them - this is a basic intuition used throughout person disambiguation approaches. Therefore we define a SPARQL rule to infer the same triples as the previous rule but this time modifying the graph pattern within the `WHERE` clause to match the name of a member of the OAK group - listed on the group's personnel page - and the name of a colleague on a separate page.

```
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
CONSTRUCT {
  ?x owl:sameAs ?y .
   ?x foaf:page ?p
}
WHERE {
  <http://oak.dcs.shef.ac.uk/people> foaf:topic ?x .
  <http://oak.dcs.shef.ac.uk/people> foaf:topic ?y .
  ?p foaf:topic ?z .
  ?p foaf:topic ?u .
  ?x foaf:name ?n .
  ?y foaf:name ?m .
  ?z foaf:name ?n .
  ?u foaf:name ?m .
  FILTER (<http://oak.dcs.shef.ac.uk/people> != ?p)
}
```

Using the above rules identifies web pages within the dcs which cite the group members and their equivalent instances from those pages. New instances of *foaf:Person* are constructed for each member of the research groups within the department. For each group member we take the instances of *foaf:Person* and assign the information from the instance description to the new *foaf:Person* instance. This *fuses* the data from separate instances to provide a richer description. Also for each group member we assign each page where an equivalent instance was found and relate this page to the new *foaf:Person* instance using *foaf:page*. When the instance is dereferenced this will provide links to all the web pages which cites the person. For each group member we create a new minted URI according to "*Cool URIs for the Semantic Web*"[5]. We use the same person namespace as for the temporary URIs but with the person name as it appears on the group personnel page (with titles removed) and append this to the namespace to produce a URI for the DCS member (e.g. `<http://data.dcs.shef.ac.uk/person/Matthew-Rowe>`).

5.3 Assigning People to Publications

Our *linked dataset* now contains instances of *foaf:Group* and *foaf:Person* describing research groups and their members. We must now identify publications which have been written by the group members. We implement a basic strategy of name matching using an abbreviated form of the names of the group members. For instance for the name "Matthew Rowe" we break down the name into several citation formats: "M Rowe", "Rowe M", "M. Rowe". The publication database has no single strategy for naming and therefore several different formats must be accounted for. It is worth noting the imprecision such a strategy would lead to if it was applied when interlinking data. However in this context it is applicable to use such a technique given the localised context of the publication database - as it only stores publications by members of the department. Using the above example we formulate queries based on several name abbreviations in order to match a group member with the publications he/she has written. An example rule is as follows:

```
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
CONSTRUCT {
  <http://data.dcs.shef.ac.uk/person/Matthew-Rowe>
    foaf:made ?p
}
WHERE {
  ?p rdf:type bib:Entry .
```

[5]http://www.w3.org/TR/2007/WD-cooluris-20071217/#cooluris

```
  ?p foaf:maker ?x .
  ?x foaf:name ?n
  FILTER regex(?n, "M.*Rowe", "i")
}
```

This rule finds an instance of *bib:Entry* which has an author whose name matches the above regular expression. The inferred triple then constructs a relation between the group member and the publication using *foaf:made* - indicating that the paper was produced by the person. For each paper that is found to have been authored by a group member we place the paper and its description within the *linked dataset*. We maintain the same URI as before (containing the paper namespace and the increment of the paper count).

Figure 3 shows a snippet of the compiled dataset. By enriching data with formal semantics - where the data is leveraged from heterogeneous sources - we are provided with a rich interpretation of legacy data. This allows SPARQL queries to be performed over the dataset in order to extract knowledge - this was previously limited without a large amount of manual processing. For instance we can ask for all the groups have have worked together on papers and what were the papers called.

6. LINKING TO THE WEB OF LINKED DATA

At this stage in our approach we have extracted legacy data as triples and have built an interlinked dataset describing people within the DCS, their publications and the research groups they are members of. This dataset must now be linked into the Web of Data to provide relations with equivalent resources and related information in distributed datasets. The advantage of this - from the perspective of members of the DCS - is that once equivalent person instances are found within external bibliography databases then all the papers written by that person, and do not appear on the DCS publication database, will be provided by looking up the URI of the DCS member.

According to [8] author disambiguation is one of the common problems faced by the linked data community. In certain cases, wrongly created *owl:sameAs* links result in the incorrect collection of publications being returned when an author URI is looked up. For now we implement a conservation strategy to link members of the DCS with publications which they have authored which are contained within external datasets. We use a similar person co-occurence strategy as when detecting equivalent *foaf:Person* instances previously. We assume that a person will author a paper with the same people that they work with. We construct a SPARQL rules which uses the notion of a networked graph [12] to query the DBLP linked dataset[6]. The rule works as follows:

```
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX dc:<http://purl.org/dc/terms/>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
CONSTRUCT {
  ?q foaf:made ?paper .
  ?p foaf:made ?paper .
  ?q owl:sameAs ?x .
  ?p owl:sameAs ?y
}
WHERE {
  ?group foaf:member ?q .
  ?group foaf:member ?p .
  ?q foaf:name ?n .
  ?p foaf:name ?c .
  GRAPH <http://www4.wiwiss.fu-berlin.de/dblp/>
  {
    ?paper dc:creator ?x .
    ?x foaf:name ?n .
    ?paper dc:creator ?y .
    ?y foaf:name ?c .
  }
  FILTER (?p != ?q)
}
```

For each group within the *linked dataset* the above SPARQL rule gathers all the group members and checks their names against the networked graph for publications where those people have worked together. The URI of the paper which matches the query is then assigned to the group members in using the *foaf:made* relation. The authors of the paper within the DBLP dataset are also detected as referring to the group members and are associated to those *foaf:Person* instances using *owl:sameAs*. Using such a query produces the following relations.

```
<http://data.dcs.shef.ac.uk/person/Fabio-Ciravegna>
  owl:sameAs
    <http://www4.wiwiss.fu-berlin.de/dblp/resource/person/169384> ;
  foaf:made
    <http://www4.wiwiss.fu-berlin.de/dblp/resource/record/conf/icml
      /IresonCCFKL05>
  foaf:made
    <http://www4.wiwiss.fu-berlin.de/dblp/resource/record/conf/ijcai
      /BrewsterCW01>
```

In order to expose linked data we have deployed our dataset using static RDF files according to Recipe 1 from "How to Publish Linked Data"[7] and Recipe 2 for slash namespaces from the "Best Practices for Publishing RDF vocabularies"[8]. This serves our purpose as URIs are dereferenceable in our published data and will allow this deployment to be upgraded to more advanced setups, such as Drupal, in the near future without the URIs returning a 404 response.

7. CONCLUSIONS

This paper has presented an approach currently in use to convert legacy data to linked data. The paper places emphasis on the first stage of the process *triplification* of legacy data as this has been the most thoroughly investigated portion of the work. We believe that the results from the evaluation demonstrate the effectiveness of using trained Hidden Markov Models to extract legacy data from HTML documents and RSS feeds. Although we have used such techniques to extract person information, the approach could be applied to other domains in which legacy data is locked away within HTML documents and devoid of machine-processable markup. In such cases the HMMs would be trained for the specific information which is to be extracted. The triplification and coreference resolution stages of the approach have provided a stable testbed on which we plan to explore statistical methods for interlinking our dataset into the Web of Linked Data. Our future work will investigate such methods in order to contribute to the Linked Data community. Once

[6]http://www4.wiwiss.fu-berlin.de/dblp/

[7]http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/

[8]http://www.w3.org/TR/swbp-vocab-pub/

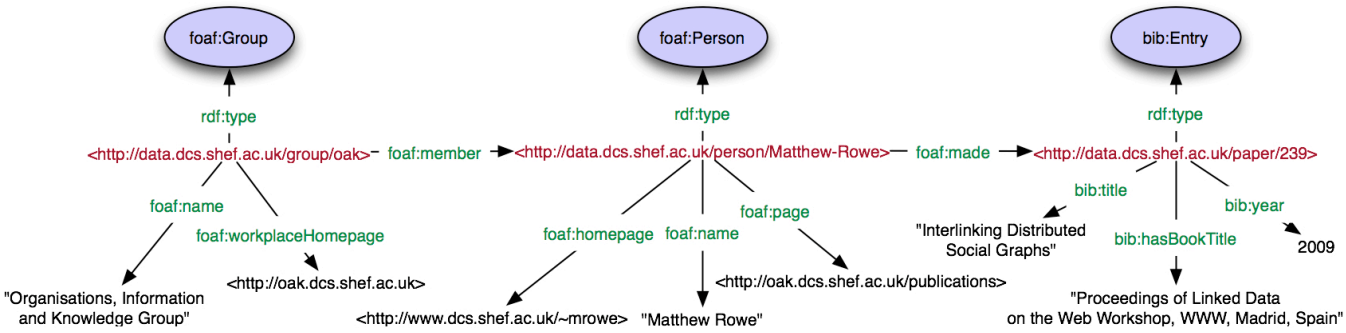

Figure 3: A snippet of the interlinked dataset following coreference resolution

we have linked our *linked dataset* to additional datasets then we plan also provide VoiD descriptions [1] of those links to enable easier consumption of the data. At present our top level components in the produced dataset are the research groups. We plan to use this project as the blueprint for producing linked data from all departments and faculties in the university, described using the Academic Institution Internal Structure Ontology[9].

8. REFERENCES

[1] K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. Describing Linked Datasets - On the Design and Usage of voiD, the 'Vocabulary of Interlinked Datasets'. In *WWW 2009 Workshop: Linked Data on the Web (LDOW2009)*, Madrid, Spain, 2009.

[2] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumüller. Triplify light-weight linked data publication from relational databases. In *18th International World Wide Web Conference (WWW2009)*, April 2009.

[3] F. Ciravegna, S. Chapman, A. Dingli, and Y. Wilks. Learning to harvest information for the semantic web. In *Proceedings of the 1st European Semantic Web Symposium (ESWS-2004)*, May 2004.

[4] P. Coetzee, T. Heath, and E. Motta. Sparqplug: Generating linked data from legacy html, sparql and the dom. In *Linked Data on the Web (LDOW2008)*, 2008.

[5] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[6] D. Freitag and A. K. Mccallum. Information extraction with hmms and shrinkage. In *In Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*, pages 31–36, 1999.

[7] E. Hetzner. A simple method for citation metadata extraction using hidden markov models. In *JCDL '08: Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries*, pages 280–284, New York, NY, USA, 2008. ACM.

[8] A. Jaffri, H. Glaser, and I. Millard. Uri disambiguation in the context of linked data. In *Linked Data on the Web (LDOW2008)*, 2008.

[9] M. Lan, Y. Z. Zhang, Y. Lu, J. Su, and C. L. Tan. Which who are they? people attribute extraction and disambiguation in web search results. In *2nd Web People Search Evaluation Workshop (WePS 2009), 18th WWW Conference*, 2009.

[10] K. Möller, T. Heath, S. Handschuh, and J. Domingue. Recipes for semantic web dog food - the eswc and iswc metadata projects. In *6th International and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007)*, pages 795–808, November 2007.

[11] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical report, W3C, 2006.

[12] S. Schenk and S. Staab. Networked graphs: a declarative mechanism for sparql rules, sparql views and rdf data integration on the web. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 585–594, New York, NY, USA, 2008. ACM.

[13] L. Shi, D. Berrueta, S. Fernandez, L. Polo, and S. Ferna?dez. Smushing rdf instances: are alice and bob the same open source developer? In *ISWC2008 workshop on Personal Identification and Collaborations: Knowledge Mediation and Extraction (PICKME 2008)*, October 2008.

[14] D. Thamvijit, H. Chanlekha, C. Sirigayon, T. Permpool, and A. Kawtrakul. Person information extraction from the web. In *6th Symposium on 6th Symposium of Natural Language Processing*, 2005.

[15] X. Wan, J. Gao, M. Li, and B. Ding. Person resolution in person search results: Webhawk. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 163–170, New York, NY, USA, 2005. ACM.

[16] K. Watanabe, D. Bollegala, Y. Matsuo, and M. Ishizuka. A two-step approach to extracting attributes for people on the web. In *2nd Web People Search Evaluation Workshop (WePS 2009), 18th WWW Conference*, 2009.

[17] B. Zhou, W. Liu, Y. Yang, W. Wang, and M. Zhang. Effective metadata extraction from irregularly structured web content. Technical report, HP Laboratories, 2008.

[18] J. Zou, D. Le, and G. R. Thoma. Structure and content analysis for html medical articles: a hidden markov model approach. In *DocEng '07: Proceedings of the 2007 ACM symposium on Document engineering*, pages 199–201, New York, NY, USA, 2007. ACM.

[9]http://vocab.org/aiiso/schema#