Extracting and Re-using Structured Data from Wikis

Jonathan Isbell and Mark H. Butler
Digital Media Systems Laboratory
Bristol

HPL-2007-182

14th November, 2007*

wikipedia; semantic web; information extraction; wikis; metadata

This report investigates simplifying the creation of structured data for use in Semantic Web applications. In the first phase of work, a prototype is created that extracts structured data on companies and unstructured data on acquisitions from Wikipedia. It then reuses this information in a data browser that can provide faceted, map and timeline views. In the second phase, we investigate more generic approaches for extracting structured data and related schema information from Wikipedia. We use this information to create user interfaces that simplify the creation of structured data about related topics. This demonstrates that it is possible to simplify the creation and re-use of structured data in ways that benefit users.

* Internal Accession Date Only

Extracting and Re-using Structured Data from Wikis

Jonathan Isbell jon@isbell.net
Mark H. Butler mark.butler2@hp.com

Digital Media Systems Department
HP Labs, Bristol

Abstract: This report investigates simplifying the creation of structured data for use in Semantic Web applications. In the first phase of work, a prototype is created that extracts structured data on companies and unstructured data on acquisitions from Wikipedia. It then reuses this information in a data browser that can provide faceted, map and timeline views. In the second phase, we investigate more generic approaches for extracting structured data and related schema information from Wikipedia. We use this information to create user interfaces that simplify the creation of structured data about related topics. This demonstrates that it is possible to simplify the creation and re-use of structured data in ways that benefit users.

Keywords: Wikipedia, Semantic Web, Information Extraction, Wikis, Metadata

Acknowledgements: Jonathan Isbell, a student from Bristol University, carried out the work described here during a three month internship at HP Labs Bristol. Many thanks to Georgi Kobilarov of the DBPedia project for feedback on an earlier version of this report.

1 Introduction

1.1 Context

Often we draw a distinction between *unstructured data*, such as text or web pages, and *structured data* that has been through a data modeling process, for example data in a relational database [Grimes2005]. We also draw a distinction between *data* and *metadata*, where the latter is commonly defined as data about data, for example the name of the author of an electronic document [Good2002].

There is an increasing need for everyday users to be able to create metadata and structured data. For example, they may be contributing information to a very large collection of information, such as an item description on Ebay, so it must be in a consistent format in order to be found. Alternatively, the user may need to describe an information resource that is not textual so is not searchable using free text search, such as photos, video or audio.

[Doctorow2001] gives six reasons why metadata is often inaccurate or unreliable, and we propose they are relevant to structured data as well:

1. Metadata creation can be too difficult or time consuming to produce accurately.
2. People may seek to create advantage by providing unreliable data.
3. Metadata creators are often not sufficiently impartial.
4. The schemas used to capture the data are often not neutral.
5. Measurements cannot completely capture all facets of reality.
6. There are often multiple ways to describe or classify things.

Although we do not agree with [Doctorow2001] that these problems present "*insurmountable obstacles*" we do think they are worthy of investigation. As structured data is modeled data, when everyday users create structured data they are performing some level of data modeling. This is a conceptually hard problem, so it is not surprising they can find it difficult.

In this report, we describe work that investigates an approach to simplifying the creation of structured data. The work is divided into two phases: the first phase involves building a domain specific prototype, and the second phase involves investigating a more generic approach.

1.2 Phase 1: Domain specific prototype

In the first phase we investigate whether it is possible to take information in Wikipedia, stored in both structured and unstructured forms, and then transform that information into a reusable structured form such as RDF [Manola2004]. We create a prototype that

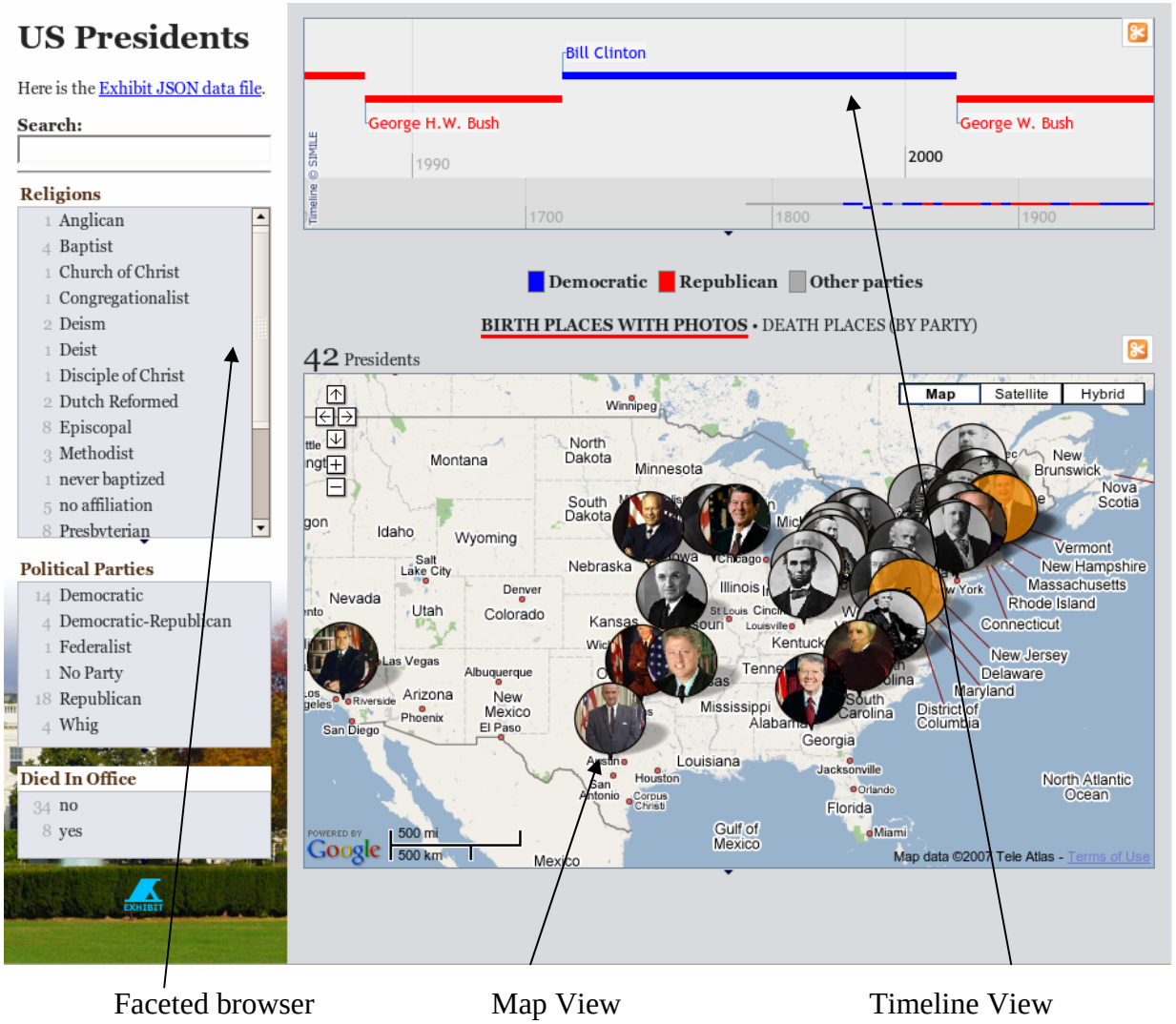

Figure 1 Screenshot of US Presidents Exhibit

extracts information on technology companies and their corporate acquisitions. In Wikipedia, general information about the company is available in structured form whereas acquisition information is only available as free text i.e. unstructured form.

Once we have extracted this information, we will then present it in a summarized form using a faceted browser called Exhibit [Huynh2007a]. This allows the filtering of information based on properties, as well as timeline or map views. An example of the views possible in Exhibit is shown in Figure 1.

Our primary aim in this phase is to demonstrate an approach for non-technical users to create structured data in a standard, reusable form without having to learn any new technical skills. Wiki software is an ideal platform because it is easy for non-technical users to learn and many organizations are already using this technology.

The secondary aim is to demonstrate the potential value of making data available in structured form by providing the user with some alternative ways of viewing and exploring the data. The approach outlined here is equally applicable to Wikis used within the enterprise domain, where providing summary views would allow project leads to analyze documentation and data created by their team and so make better decisions.

1.3 Phase 2: Generic Approaches

In the second phase, we explore generic ways to extract all types of structured information in Wikipedia. We investigate how this extracted information can be used to simplify the task of creating additional structured data rather than just reusing data that has already been created. The aim is

to address three problems which contribute to the difficulty of creating structured data.

The first problem is that for structured data to be usable, users need to adopt a common approach to encoding the data. Adopting a common vocabulary when a system is under decentralized control can be difficult, because the separate parts of the system all tend to develop their own idiosyncratic vocabulary.

The second problem is that data representation formats that support a wide range of vocabularies often require the use of disambiguating mechanisms such as namespaces in order to distinguish between these different vocabularies. Often namespaces are based on URIs, which increase complexity as they are long and not always easy to remember. Some formats support namespace prefixing i.e. the namespace only need to be included once and is subsequently referred to using a prefix. Even with prefixing, namespaces drastically increase complexity for some users.

The third problem is that we need to that ensure people encode concepts consistently, for example to avoid several variations of a company name. One way to avoid a proliferation of these variations is to list all the names in a controlled vocabulary, so we effectively agree on a single variation, and misspellings are rejected at data entry time.

In our prototype we address these three issues as follows: we avoid users having to formulate how to describe a particular item, because we give them a form to prompt them to enter appropriate information. We do not require the user to remember namespaces, although we do use namespaces internally. We also automatically process existing metadata to create a vocabulary to guide value data entry where appropriate. Systems such as Ebay which depend on users entering metadata use some of these approaches. Another similar system is Freebase, a system being developed by Metaweb technologies [O'Reilly2007]:

"*Freebase [is] a website that sits on top of a new kind of database. Just as Wikipedia lets people contribute information to its articles, Freebase will let anybody contribute, correct or recombine data. The difference is that information on Wikipedia tends to be "unstructured"—i.e., buried in text—whereas on Freebase it will be structured, so that each item can be re-used in any context."* [Economist2007]

Unlike Ebay, the system presented here gives users full control over what metadata is entered rather than having to conform to a predetermined data model. In contrast to Freebase, we propose a scheme which will work with existing Wiki platforms rather than requiring a new platform. In addition we use RDF to ensure that the structured metadata is created in a format that can be reused in a number of different ways.

1.4 Structure of Report

The remainder of this report is structured as follows: Section Two describes Phase One, specifically how unstructured and structured data is represented in Wikipedia, how we extracted information from templates, how we converted this to RDF and how we viewed it using Exhibit. Section Three describes observations and further work based on Phase One. Section Four details Phase Two which investigated more generic extraction techniques that produce schema information as well as instance data. It also investigated how this information could be reused to create better user interfaces. Section Five describes observations and future work based on Phase Two. Section six then finishes with conclusions.

2 Phase One: Domain Specific Prototype

2.1 Wikipedia

Wikipedia [Wikipedia2007] is one commonly consulted information resource on the Internet, although there is some controversy about its authority [Giles2005], [Britannica2006]. Originally Wikipedia contained solely unstructured information, but it is now incorporating structured information for certain types of topics using *templates*. For example, entries that describe companies may provide information such as the name of the company, founder(s) and chief executive officer in structured form as shown in Figure 2. Currently these templates are primarily used to ensure consistent formatting between different topics in the same class.

2.2 Structured Data in Wikipedia

2.2.1 WikiText

Both the unstructured and structured data in Wikipedia is represented in a format called WikiText. This format was deliberately created to be simpler to edit than comparable formats for representing human readable information such as HTML. Here is an example of typical WikiText markup:

```
== heading ==
this is a paragraph
* this is a list item
* this is another list item
 [[Wikipedia|link to article]]
```

2.2.2 Templates

MediaWiki, the software platform behind Wikipedia, introduced a feature in Version 1.2.6 called templates that provides a way of representing certain pieces of information that were common to a number of different entries. Templates allow for the standardized presentation of data for every instance of a template.

Figure 2 - Display of typical Infobox template in Wikipedia

Articles which can be categorized into a certain type e.g. a film or place often make use of special MediaWiki templates called an *Infobox Template*. An Infobox Template contains structured data about the subject being discussed by an article. Templates not used as Infoboxes are used for adding extra information to a piece of text, for example linking to coordinates on a map, or in a series of articles to give links to related articles in a topic area, so that each topic includes the same set of links.

Templates simplify the task of extracting structured data and transforming it into other formats because the data is in a consistent, easy to parse structured format. The general format for templates is as follows:

```
{{ TemplateName
| field1 = value1
| field2 = value2
| field3 = value3
}}
```

TemplateName is the name of the template to use when reformatting. Field1 to Field3 are the names of fields that exist in the template and value1 to value3 are the values to assign to the fields.

Here is an excerpt from the template that generates the output shown in Figure 2:

```
{{Infobox_Company |
| company_name = Hewlett-Packard
| company_logo = [[Image:Hewlett-...
| company_type = [[Public company|...
| slogan = Invent.
| foundation = [[Palo Alto, Califo...
| location_city = Palo Alto, Calif...
| location_country = USA
| key_people = [[William Reddingto...
| num_employees = 156,000 (2007)
| industry = [[Computer Systems]]...
| products = [[Calculators]] [[C...
| revenue = {{profit}}$91.7 billio...
| net_income = {{profit}}$6.2 bill...
}}
```

In addition to Infobox templates, Pagelinks are another potential source of structured information in Wikipedia. We will discuss some ways of using Pagelink data later in the report.

2.2.3 Extracting Structured Data

Several different extraction methods were investigated to get structured data from templates.

First, we investigated a web scraping approach to retrieve articles from the Wiki in the same way as HTML viewed by the user. This approach does not work well because information is lost between the WikiText version of the template and the HTML version.

Second, we tried converting the raw WikiText into an Abstract Syntax Tree (AST). This method avoided some of the disadvantages of scraping HTML such as unneeded presentational markup around templates. However it was not possible to reuse an existing WikiText parser as all existing parsers convert WikiText directly to HTML rather than supporting an intermediate stage, so it was necessary to implement a simple parser from scratch. Due to time constraints it was not possible to write a parser that supported the entirety of MediaWiki WikiText syntax.

The method finally chosen uses regular expressions to extract information. This means the required information can be extracted from the page independently of the rest of the structure of the page.

One limitation of the Java implementation of Regular Expressions is that it does not support nested or recursive expressions. This made it difficult to match nested tokens such as when templates are nested within templates as shown below:

```
{{TemplateName
| field1 = value1
| field2 = value2
| field3 = {{AnotherTemplate |
anotherfield1 = anothervalue1 }} value3
}}
```

At first we attempted to use regular expressions which were written to include a fixed number of recursion levels. However this means the length of the expression grows and becomes unmaintainable as the number of levels of recursion increases. This problem was overcome by writing a Java method for dealing with matching nested opening and closing tokens which allowed bracket matching to an infinite level.

In the prototype the structured template extraction was not totally generic as special stripping was required on different fields of the template to remove unrequired WikiText syntax. For example the profit field used a different syntax and formatting to presenting the information.

The work described here is not the first attempt to extract structured information from Wikipedia. This has already been performed by the DBPedia project [DBPedia], [Auer2007]. For this work, we decided not to use DBPedia, because we wanted to get a better understanding of the extraction process. This was in order to explore how these techniques could be applied to general Wiki content, rather than just Wikipedia. However an alternative approach would have been to use the DBPedia dataset. There will be more discussion on the relationship between this work and DBPedia later in the report.

2.2.4 Limitations of Wikipedia Structured Data

As already mentioned, templates in MediaWiki were designed to support the common formatting of data in related topics. Therefore, when using them to extract data, a number of potential problems become apparent: first, different authors do not encode values the same way, even though the information is about the same type of thing. For example the profit field of the company template could include the actual profit, an icon to show whether a profit or loss was made, the year the data was collected from or a reference to the source of the data. There are standard ways to write certain properties such as dates and currency in Wikipedia but even when there are standards there are no way to enforce them.

Second, there are variations in the property names used on different templates. For example in the template about companies, both `company_name` and `name` are used, because `company_name` has been deprecated in favor of `name`.

This is related to the third problem, which is there is no standard way to denote the deprecation of a field within a template. This often means templates will contain unnecessary markup within template design. This means Wikipedia's Company Infobox has to check the existence of both `company_name` and `name`.

Despite these complexities, extracting template data is much easier than unstructured data, which will be discussed later.

2.3 Retrieving Articles from Wikipedia

Because we were interested in extracting structured information represented in templates, our prototype needed to work on the underlying WikiText rather than the HTML presented to the user. MediaWiki offers three HTTP interfaces for extracting data apart from the HTML view presented to users. The first way is called `Special:Export` which allows the exporting of entire categories. Articles are returned as WikiText within an XML container. The second way is to append `?&action=raw` to the URL of any article in order to obtain the WikiText. The third way is using the Mediawiki API which allows more specific retrieval of articles such as different revisions and metadata about the article.

We decided the second way was the best approach to retrieve individual pages, as the article would be in its original WikiText form. The prototype can retrieve articles listed in a file or from a specific Wiki category. It also uses the `Special:Export` feature to get a list of all articles within a category.

Because we were extracting the information from a local replica of Wikipedia, it was possible to create retrieval lists by placing articles within specific categories. This made it easy to use the software as then it was just a case of tagging the relevant articles so that it would be included in the extraction process.

It would have been more difficult to do this on the primary version of Wikipedia as live articles would need to be changed. However this process of cleaning or fixing errors in the data which only become apparent when it is treated as structured data is common in data extraction and reuse.

In the prototype, conversion from the Wiki to RDF was done on an on-demand basis. This was only practical because the prototype was working on a limited set of topics. More scalable approaches are discussed in Section 3.2.3.

2.4 Unstructured Data in Wikipedia

As previously explained there is a lot of information within Wikipedia about related topics which does not have a consistent structure. Sometimes the information is written as free text, but other times it is structured in some way.

The acquisition data, like the classified advert data investigated in [Soderland1999], is semi-structured text that may be ungrammatical and is represented in a tabular or abbreviated form. Although this data does have some structure, different authors will have represented the data in different ways, so it is necessary to check that any algorithm supports these multiple representations. For example, information about acquisitions by technology companies is formatted using tables, lists, paragraphs and links to other articles.

Here is an example of the tables approach:

```
| [[June 5]], [[2007]]
| [[PeakStream]]
| Parallel Processing
| '''undisclosed'''
|<ref
name="PeakStream">"[http://www.theregis
ter.com/2007/06/05/google_buys_...
```

And here is an example of the list approach:

```
*'''Opsware''': On [[July 23]]
[[2007]], HP announced it was going to
acquire [[Opsware]], a developer of
data centre management systems, for
$1.6 billion<ref>{{cite news |title= HP
buys Opsware for $1.6 billion
|url=http://edition.cnn.com/2007/TECH/b
iztech/07/23/opsware.hp.reut/
|work=[[CNN]] |date=2007-07-23
|accessdate=2007-07-23 }}</ref>.
```

Semi-structured data of this type is not amenable to natural language processing methods that try to analyze sentence structure, for example identifying verbs, nouns etc. Instead it is necessary to use information extraction techniques [Grisham1997], [Manning2005], [Cunningham2005] such as list lookup extraction, fillers and spatial / proximity analysis. Unfortunately, because of the variations in representation, a certain amount of hand tailoring of the algorithms was necessary. There has been work on assisting users to create extractors for these tasks [Kuhlins2002], [Soderland1999], but we decided adaptive assisted extraction was too complicated to investigate within the scope of the current work.

2.4.1 List Lookup extraction

One information extraction technique we used was list look up extraction. This finds and extracts names, places and classifications using simple pre-computed

lists of words known as gazetteers [Tablan2003]. For example, this method of extraction could be used to determine the industry of an acquired company using a list of technology company keywords. Here is a simple gazetteer for this purpose:

```
Print
Web
VOIP
Storage
```

By searching the text about the acquisition for any incidences of these words, it is possible to determine the type of acquisition. However this does require pre-existing, possibly manually created gazetteers.

2.4.2 Fillers

Fillers [Manning2005] involve search for known patterns using regular expressions [JavaRegex2007]. For example, we can often identify years if we make the assumption they will either be in the twentieth or twenty first century as then they will follow the pattern:

```
(19|20)\d{2}
```

Sometimes the pattern is identified by context, so it is necessary to search for a pattern before, known as a pre-filler, or after, known as a post-filler. For example a reference to a transfer of money uses both a pre and post filler so can be matched by an expression such as:

```
[\$£][\d,]+\s*(in cash|in stock)
```

2.4.3 Spatial / Proximity Analysis

Spatial / proximity analysis is where information is extracted because of its location to other extracted information in the text. For example a year, month and day are often close together in a single date. Therefore if we can locate the year and month and find another number nearby it is probably a day.

2.4.4 Extensibility

There are a number of frameworks for information extraction such as GATE [Cunningham2002]. We investigated this framework but we decided the learning curve was high so we did not use it in the work described here. Instead the prototype was designed in an extensible way so that code for dealing with different forms of acquisition data could be added with minimal changes. This was done using a Factory design pattern. Different extractors can register with the factory using Java Reflection [Sun2007] so it is simple to add new extractors. Future work, as described in Section 3.2.4, could investigate how the current prototype could be developed into a more generic framework.

2.5 Output to RDF

Once the data was extracted from Wikipedia, it was represented using RDF. RDF provides a way to express simple statements, describing resources with properties and values [Manola2004]. In RDF resources and properties are represented by globally unique URIs. Because of this global uniqueness, it is very easy to merge two pieces of RDF compared with other data modeling formats.

The Jena Framework was used to create RDF models from the company and acquisitions data extracted from Wiki articles. In addition to the standard RDF formats, it was necessary to support JavaScript Object Notation (JSON) [JSON] because it is required by the Exhibit faceted browser. JSON is an alternative lightweight data exchange format that is increasing in popularity. As the only way to get Jena to produce JSON is from SPARQL SELECT queries, it was necessary to write a custom class to output RDF models in JSON. It is important to note the conversion from RDF to JSON is inherently lossy, as it omits namespace information. [JDIL] discusses some possible approaches to overcome this. Some typical output from this custom class is shown over the page:

```
{"items": [
  {
    "logo":
"http://upload.wikimedia.org/wikipedia/
en/...",
    "label": "Hewlett-Packard",
    "type": "Company",
    "city": "Palo Alto, California",
    "country": "USA",
    "slogan": "Invent.",
    "founded": "Palo Alto, California,
California (1939)",
    "keypeople": [
      "Dave Packard, Co-founder",
      "Mark V. Hurd, Chairman, CEO and
President",
      "William Reddington Hewlett, Co-
founder"
    ],
    "netincome": "$6.2 billion United
States dollar (2006)",
    "latlng": "37.44462,-122.16077",
    "industry": [
      "Computer Systems",
      "Computer software",
      "Consultant",
      "IT Service Management",
      "Peripheral"
    ],
    "ORG": "Hewlett-Packard",
    "employees": "156,000 (2007)"
  },
  {
    "description": "On August 23 2003,
HP acquired PipeBeach...",
    "month": "August",
    "acquired": "PipeBeach",
    "year": "2003",
    "acquirer": "Hewlett-Packard",
    "label": "PipeBeach",
    "day": "23",
    "type": "Acquisition",
    "date": "2003-08-23"
  }
]}
```

2.6 Viewing with Exhibit

Exhibit [Huynh2007a] is a simple framework for publishing structured data that supports faceted browsing and multiple views of data. It does not require a database or server side programming languages as it runs on the client side using JavaScript.

Setting up Exhibit is very simple and involves including two JavaScript files: one to call the Exhibit API and the other containing the data source of RDF encoded as JSON. Separating the instance and schema JSON data sources in this way allows for easy regeneration of instance data. The output from Exhibit can then be styled and configured using HTML and XML. There are a number of tutorials [Huynh2007b] available which give details on the creation of exhibits.

In addition to a faceted view, Exhibit also made it possible to provide a timeline with the dates of acquisitions and a map view. This displayed the locations of the headquarters of the acquired and acquiring companies, generated by calling a Geocoding service to convert place names into latitude and longitude coordinates. See Figure 3 and Figure 4 for screenshots of these views on the extracted data.

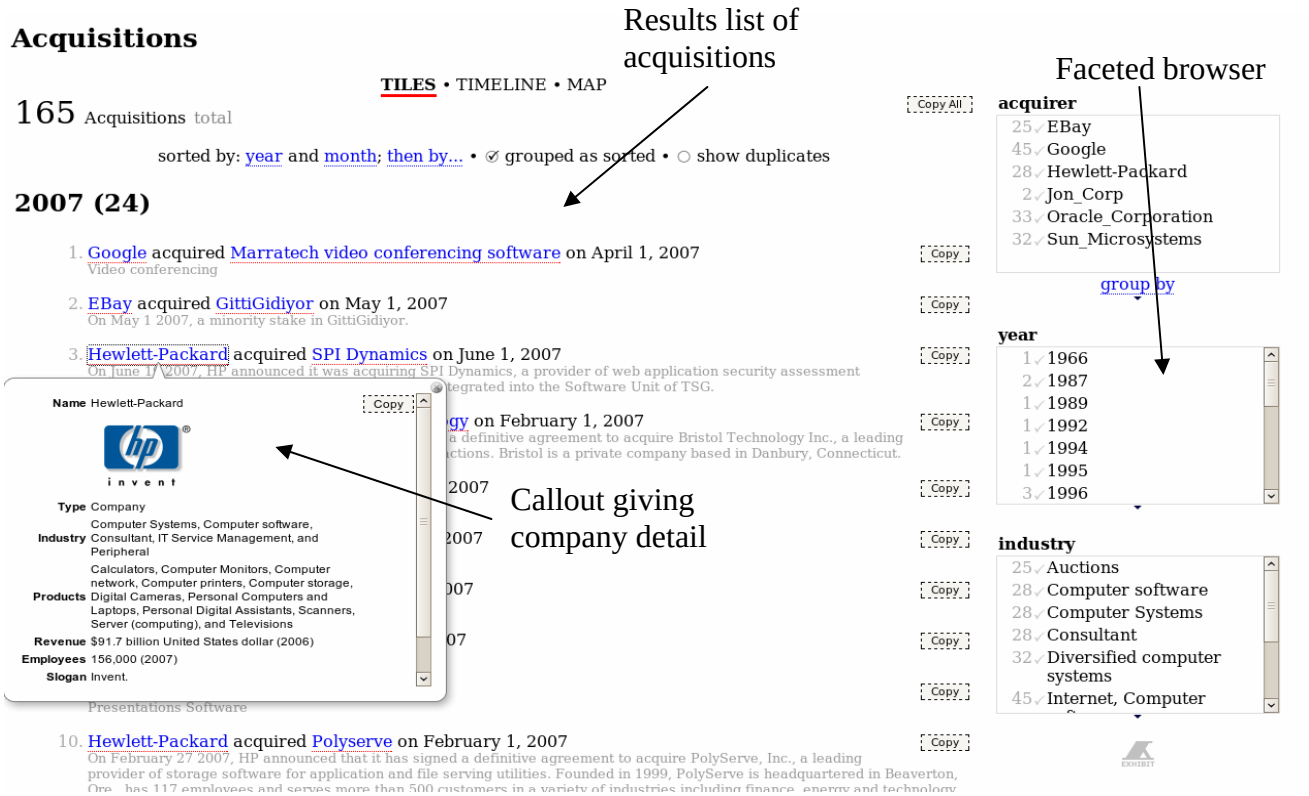

Figure 3 Prototype listing extracted acquisition information

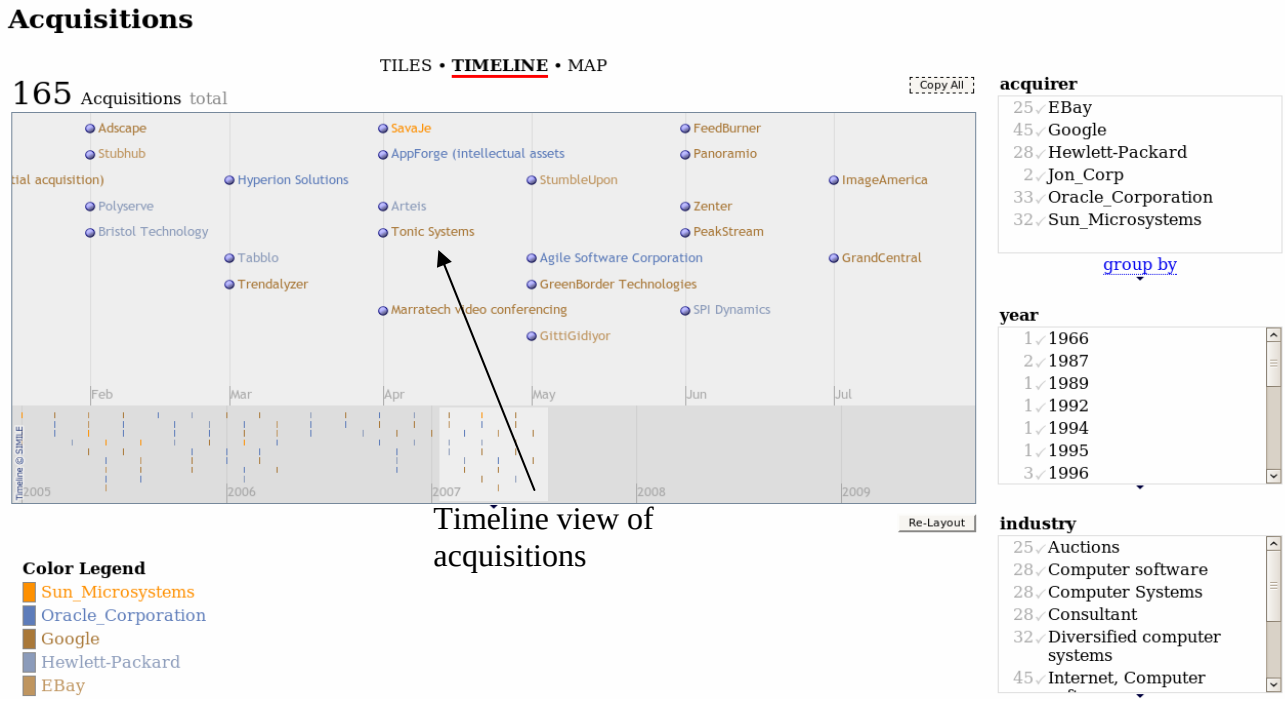

Figure 4 Prototype displaying dates of acquistions on a timeline

3 Phase One: Conclusions and Further Work

3.1 Conclusions

In summary, in Phase One we demonstrated that it is possible to extract and reuse structured data about companies and unstructured data about acquisitions from Wikipedia. Once the information was extracted as RDF, it was possible to view this in Exhibit. The prototype is able to run alongside existing MediaWiki installations without making any changes to the installation, although we did make some small changes and corrections to the data to get the system to work efficiently. Displaying data in a different format, such as being able to view the company and acquisition data as a timeline, allows the user to extract more information from the data.

Technologies such as Wikis make it increasingly easy for users to create and publish content on the Internet. This work shows by paying attention to how content is structured, it is possible to reuse that content in a variety of new ways. Even if the content was not structured at creation time, it is possible to use automatic techniques to create structured versions of the content that can be used in this way.

3.2 Further Work

We have identified a number of possible future directions for the work conducted in Phase One.

3.2.1 Increasing Use of Structured Data in Wikipedia

At the moment, templates are primarily summaries of articles. However articles often contain detail, for example the acquisitions data on companies, which does not belong in the summary, that could be browsed as structured data. Therefore the Wikipedia community should be encouraged to create templates for this type of common data, allowing common presentation, consistent structure and making it easier to extract information. This would avoid the need for the more difficult unstructured extraction we describe here.

3.2.2 Encouraging Common data formats in Wikipedia

Another key issue raised by the work here is the variety of formats used for data fields in Wikipedia such as earnings. It would be useful to implement a feature which enforces field formats within templates. This would ensure that data is written in a standard format, for example that a date field is always in the standard Wiki date format. [Auer2007] also gives some guidelines on producing better Wikipedia templates such as standardizing units and separating presentation and unneeded attributes.

3.2.3 Automating Recent Changes using RSS Feeds

Clearly it is desirable to keep the extracted information up to date with Wikipedia. One way to keep these up to date is to schedule the program to run at predetermined intervals. As the current implementation works by downloading all the articles from a list or specific category, this causes a large amount of network traffic every time the RDF graph is rebuilt, which limits how frequently it can occur.

One way to optimize this is by monitoring the RSS Feed for Recent Changes on the Wiki and only download changed articles that are part of the extracted data set. This would mean the program could be run more regularly and update automatically.

3.2.4 Framework for Extraction of Unstructured Data in WikiText

Although DBPedia provides a source of structured data extracted from Wikipedia templates, there are no similar tools aimed at free text in Wikipedia. A framework that contained tools to simplify the task of extracting information from free text in Wikipedia, perhaps based on existing frameworks such as GATE [Cunningham2002], is potentially very useful for building applications such as those described here.

4 Phase Two: Generic Approaches

In Phase Two we investigate whether the ideas developed in Phase One can be applied for generic extraction of structured data from Wikipedia Infobox templates. We explore using information in Wikipedia templates to automatically create schemas

or ontologies, in order to simplify the reuse of the extracted information. We then reuse the instance and schema data in order to create a user interface that makes it easier for users to create more data that conforms to a specific template.

4.1 Generic data and schema extraction

First, we discuss generic ways of extracting schema and instance information from Wikipedia.

4.1.1 Describing RDF vocabularies

As well as describing data, RDF can also be used to describe vocabularies, which define classes of resources and properties used to describe those resources, using a schema or ontology language such as RDF Schema (RDFS) [Manola2004] or OWL [McGuiness2004]. One important reason for creating such descriptions is they avoid hard coding information about a vocabulary into a program, making them more generic.

For example, RDFS contains constructs to specify human readable information about a vocabulary. In RDFS it is possible to specify a human readable label to use for a specific resource using `rdfs:label`, a longer textual description for a resource using, `rdfs:description`, and a way of linking resources to other sources of information elsewhere using `rdfs:seeAlso`. It is also possible to specify the type of class that may have a specific property, using `rdfs:domain`, and the type of object that a specific property can take using `rdfs:range`.

Information to create the RDFS can be extracted from Wiki template definitions. These are essentially Wiki articles prefixed with a special Wiki namespace [WikiNamespaces]. It is possible to pull out structured data from the template using regular expressions. On first glance, it looks difficult to extract information from templates as they are essentially designed to present information so there are many different ways of designing a template. Fortunately, most Wikipedia Infoboxes follow one of a number of different structures and so a brute force method can be attempted by testing regular expressions aimed at common designs against other template definitions to see if any of them match. If none of the standard Infobox formats match, it is still possible to extract the names of the fields from the templates as these are defined in the WikiText syntax (`{{{field}}}`) and also to generate a label from the field name by replacing underscores in the name with spaces and capitalizing the first letter of words within the field name. Here is an example of a Wikipedia template:

```
<includeonly>{| class="infobox vcard"
style="font-size:90%; width:23em;" |-!
class="fn n org" style="text-
align:center; font-size:120%;"
colspan="2" |
{{{name|}}}<!--deprecated:-->
{{{company_name|}}}<!----> |-
{{#if:{{{logo|}}}<!--deprecated:--
>{{{company_logo|}}}<!----> | <tr
class="logo"><td colspan="2"
style="text-align:center; padding:16px
0 16px 0;">{{{logo|}}}<!--deprecated:--
>{{{company_logo|}}}<!----></td></tr>}}
|-<tr class="note"><th style="text-
align:right; padding-
right:0.75em;">[[:Category:Types of
companies|Type]]</th><td>{{{type|}}}<!-
-deprecated:-->{{{company_type|}}}<!---
-></td></tr><!---->
{{#if:{{{locations|}}}       | <tr
class="note"><th style="text-
align:right; padding-
right:0.75em;">No. of
locations</th><td>{{{locations}}}</td><
/tr>}}<!---->
{{#if:{{{operating_income|}}}| <tr
class="note"><th style="text-
align:right; padding-
right:0.75em;">[[Earnings before
interest and taxes|Operating
income]]</th><td>
|}</includeonly><
```

Here is the RDF Schema containing information derived from this template:

```
@prefix company:
<http://en.wikidata.org/wiki/Template:I
nfobox_Company#>
@prefix rdf:
<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
@prefix rdfs:
<http://www.w3.org/2000/01/rdf-schema#>

company:company
   a rdfs:Class .

company:locations
    a            rdf:Property ;
    rdfs:domain  company:company ;
    rdfs:label   "No. of locations" .

company:operating_income
    a       rdf:Property ;
    rdfs:domain  company:company ;
    rdfs:label   "Operating income" ;
    rdfs:seeAlso
<http://en.wikipedia.org/wiki/Operating
_income> .
```

The OWL ontology language has some additional constructs. owl:`sameAs` indicates if two resources refer to the same thing. `owl:deprecatedProperty` indicates if a property has been superseded, and `owl:equivalentProperty` indicates if two properties are the same. As already noted Wiki templates definitions often contain fields which have been deprecated, although there is no formal way to indicate this within MediaWiki's WikiText syntax. Here is a one approach used in Wikipedia to indicate the depreciation of a property `company_slogan`:

```
{{#if:{{{slogan|}}} |
<tr class="note">
 <th style="text-align:right; padding-
right:0.75em;">
  [[Slogan]]

 </th>
 <td>{{{slogan}}}
  <!--deprecated:-->
   {{{company_slogan|}}}
  <!---->
  </td>
</tr>}}
```

Once a deprecated property is identified, `owl:DeprecatedProperty` could be used to indicate that a property has been deprecated and then `owl:equivalentProperty` could be used to denote that the deprecated property is equivalent to the superseding property.

4.1.2 Assigning URIs

One important part of generic extraction of both instance and schema information is assigning URIs to concepts and properties. In Wikipedia, there is a danger that different templates may use the same property name with a different meaning. For example many templates use a property called `type`, which can be used to describe entities as diverse as Pokémon characters, newspapers, New Testament Pypri, Military Units, Military Structures, Digital Cameras and Aircraft. Clearly `type` has a different meaning in these different contexts. Instead we consider other ways of assigning URIs to properties. Therefore we decided to assign a different namespace to each type of template, so that for example military unit type will have a different URI to digital camera type.

However, there are also examples of templates with properties that have the same name and do have the same meaning. For example there are three related templates, `company`, `defunct company` and `cooperative`. These templates describe similar entities, so they use the same properties. One way to identify this is using WikiLinks, as template definitions often use them to define properties. For example here the property `founder` links to the definition `Entrepreneur`:

```
{{#if:{{{founder|}}}|
<tr>
 <th style="text-align:right; Padding-
right:0.75em;">
  [[Entrepreneur|Founder]]
 </th>
 <td>{{{founder}}}</td>
</tr>}}
```

If two different templates point to the same definition, we know they refer to the same concept. Therefore instead of creating a new URI from the template and the property name, we use a URI derived from the property definition. Then if several different templates use the same property they will all use the same URI. In the generated schema we can use `rdfs:seeAlso`, to point to the human readable definition.

Rather than reusing existing URIs from Wikipedia to identify topics, and hence risk confusion about whether the URL refers to the original Wikipedia page or the extracted structured data we decided to create new URLs for the extracted data We note that it is considered best practice to choose namespaces which are under the control of the creator of the namespace, so that they will not conflict with someone else using the namespace for an alternative purpose, and this allows the possibility of retrieval of resources from the URI of the namespace. In the examples presented here we have used a fictitious `wikidata.org` domain.

Good general principles for publishing data are described in [Bizer2007]. For example, even if different URIs are adopted for the extracted data, it is possible to use content negotiation to provide human readable versions of the extracted data.

4.1.3 Redirects

Another potential problem is that Wikipedia uses redirects so that multiple URIs may redirect to a specific article so that people can find the article more easily. Using redirects is not ideal as according [Jacobs2004] it is better to have one URI for a resource. One approach to dealing with these redirects in RDF is to dereference a page in Wikipedia until we find the resource it corresponds to, then represent the relationship between the original URI and the target URI using `owl:sameAs` to indicate that the two URIs are equivalent. However, this means the resulting data needs to be processed by an inference engine, in order to add extra triples so that the redirected URIs return the same data as the target URI. The DBPedia project considered this approach and decided against using it, because they note the targeted URI is a preferred term, whereas the original URI is non-preferred term, and using `owl:sameAs` does not convey this information. However, we note this means it can be more difficult to find data, as then URIs that appear to work in Wikipedia do not work in the extracted data, so here we have decided to take a different approach to DBPedia and implement redirects.

4.1.4 Data Extraction

We used the same approach as described in Phase One to retrieve articles from the Wiki over HTTP. There are also a number of templates which are used within some Infoboxes to reformat data. One example of this is `Birth_date_and_age` template which converts a date in a non-standard format to the standard Wiki date format and calculates an age. Once the articles have been retrieved, a token matching function is used to extract template instances from the article to deal with these nested templates.

Then several regular expressions are used to extract the structured information from the template as before. Once the RDF representation of the template instance has been created, it is added to the Jena RDF store.

Here is an example of a template instance about England:

```
{{Infobox Country or territory
|area             = 130,395
|calling_code     = 44
|capital = [[London]](''[[de facto]]'')
|common_name      = England
|currency       = [[Pound sterling]]
|GDP_nominal      = $2.2 trillion
|prime_minister   = Gordon Brown
|sovereignty_type = Unified
|symbol_type     = Royal Coat of Arms
|time_zone        = GMT
|utc_offset       = 0
|time_zone_DST    = BST
|utc_offset_DST   = +1
}}
```

Here is the RDF, written in N3 format, extracted from that template.

```
@prefix :        <#> .
@prefix wiki:
<http://en.wikipedia.org/wiki/> .
@prefix wikidata:
<http://en.wikidata.org/data/> .
@prefix country_or_territory:
<http://en.wikidata.org/wiki/Template:I
nfobox_Country_or_territory#> .

wikidata:England
  rdfs:seeAlso wiki:England ;
  country_or_territory:area
       "130,395" ;
  country_or_territory:calling_code
       "44" ;
  country_or_territory:capital
       "[[London]] ([[de facto]])" ;
  country_or_territory:common_name
       "England" ;
  country_or_territory:currency
       wikidata:Pound_sterling ;
  country_or_territory:currency_code
       "GBP" ;
  country_or_territory:gdp_nominal
       "$2.2 trillion" ;
  country_or_territory:prime_minister
       "Gordon Brown" ;
country_or_territory:sovereignty_type
       "Unified" ;
  country_or_territory:symbol_type
       "Royal Coat of Arms" ;
  country_or_territory:time_zone
       "GMT" ;
  country_or_territory:time_zone_dst
       "BST" ;
  country_or_territory:utc_offset
       "0" ;
  country_or_territory:utc_offset_dst
       "+1" .

wikidata:Pound_sterling
  rdfs:seeAlso wiki:Pound_sterling ;
  rdfs:label "Pound Sterling" .

wikidata:England
  country_or_territory:capital
        wikidata:London .

wikidata:London
  rdfs:seeAlso wiki:London ;
  rdfs:label   "London"
```

4.1.5 Lists

A large number of the fields within Wiki templates contain lists of items, for example:

```
products =
[[Calculators]]<br />
[[Computer Monitor]]s<br />
[[Digital Camera]]s<br />
[[Computer network|Networking]]<br />
[[Personal Computer]]s and
[[Laptop]]s<br />
[[Personal Digital Assistant]]s<br />
[[Computer printer|Printer]]s<br />
[[Scanner]]s<br />
[[Server (computing)|Servers]]<br />
[[Computer storage|Storage]]<br />
[[Television]]s<br />|
```

Converting the field into a list of separate items makes the data more structured and useful for the suggestion feature of the UI. However there is no standard way within Wikipedia for encoding list items, so this is largely determined by the preference of the editor. Common list separators include “,”, “;” and “
”. Currently the list is just split when a
 is encountered within the field.

A possible algorithm here would be to count the number of occurrences of each type of list separator, choose the most frequently occurring list separator. If it has more occurrences than a minimum threshold then the field is split into a list using that token.

In some cases, such as the list of key people at Google, each list item contains a *composite value* i.e. it contains a number of values without using template markup to indicate their meaning. In the case of the Google page, the list contains not only the person’s name but also their role. Mixing different types of values in this way without some kind of structure makes it very difficult to extract the information automatically, so we have not yet identified a generic solution to this problem.

4.1.6 Data types

Another issue to consider when extracting information from the template instances is that although there are standards within Wikipedia denoting how different types of data should be formatted, for example dates or money, there is no enforcement of this and no way of denoting that a field should use this format. As RDF supports XML data typing, it would be helpful to get the data type of

the extracted values corrected. However due to the ambiguity in the Wikipedia data, it is hard to present data in its correct XML data type, so in the current implementation all fields are treated as either string literals or lists of string literals.

4.2 User Interface

The UI component allows non-technical users to interact with the prototype and presents them with a visual interface to guide them in creating structured data.

As in Phase One, a prototype of the system was implemented using Java, MediaWiki [Mediawiki] and the Jena framework [Jena]. Apache Tomcat was used as a JSP server to allow the prototype to interact with from a web browser.

A Firefox browser plug-in called GreaseMonkey [GreaseMonkey] and a JavaScript implementation of an AutoSuggest widget [Kepley2005] were used to provide client-side interaction within the browser. GreaseMonkey allows client-side changes to be made to web pages on-the-fly using JavaScript. It was possible to create a GreaseMonkey script which added a button to the toolbar on any MediaWiki editing page by manipulating the DOM of the page. This allows us to extend MediaWiki without having to change the MediaWiki code. Currently the user has to install a Firefox extension GreaseMonkey and our JavaScript file, although there are ways around this such as using a compiler which turns the GreaseMonkey script into a Firefox extension.

The button is linked to our external service which creates an overlaid form as shown in Figure 5. The overlay is populated with input elements using names and labels retrieved from the RDF model maintained by the servlet. If the template schema has not been previously extracted from the Wiki, then the servlet extracts it. Figure 6 shows all the interactions made with the user interface.

As the web service is hosted on a different domain to that of Wikipedia, it is difficult to make standard AJAX [Garret2005] requests using XMLHttpRequest as they are restricted to the same host. This is caused by a security restriction called *Same Origin Policy* implemented in all modern browsers. One way to overcome this is to use the "*dynamic script tag*" method, where `<script>` tags are inserted into the DOM and the `src` attribute is pointed at a JSON based web service [Herrington2006].

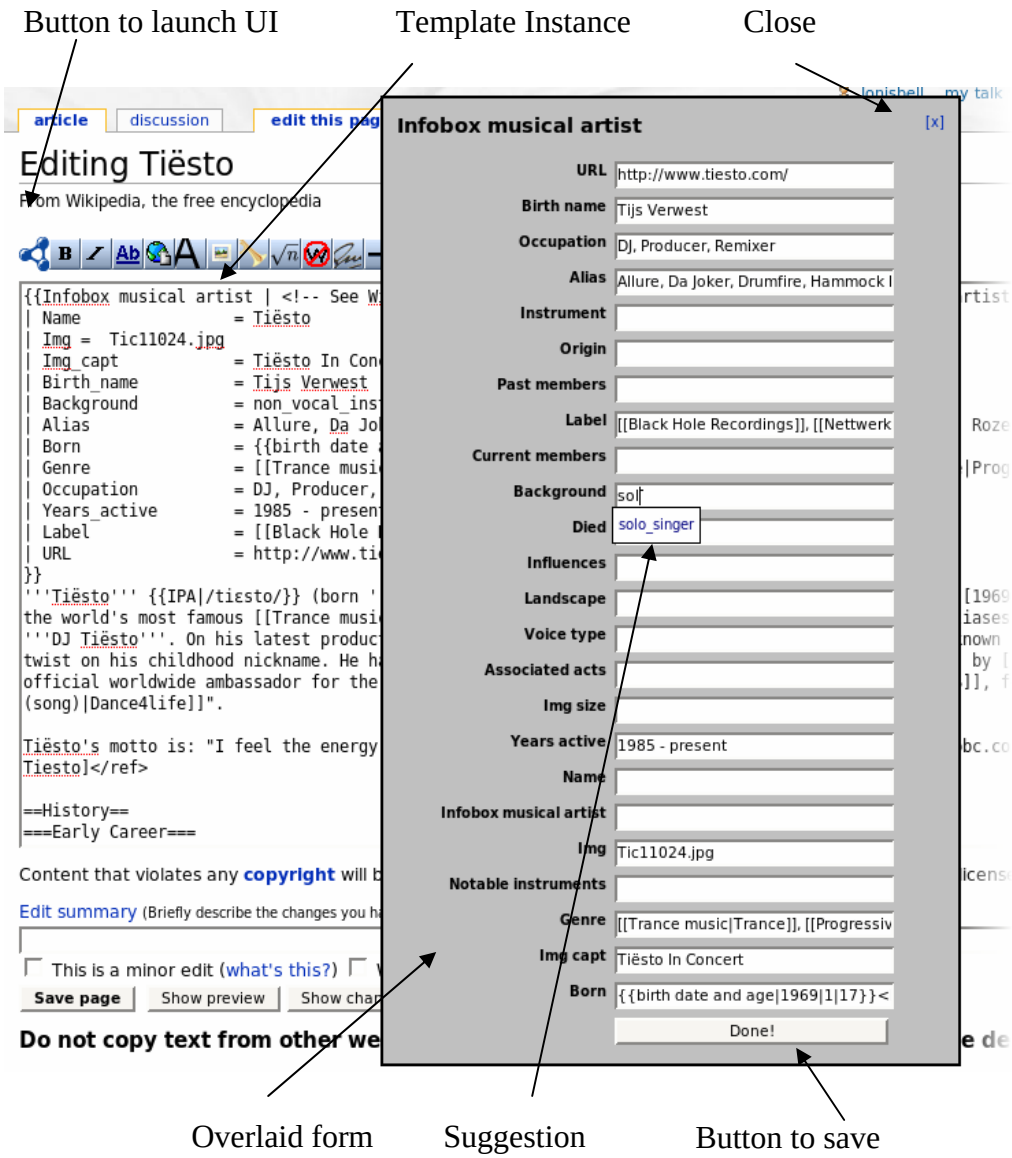

Figure 5 - Screenshot of suggest user interface

4.2.1 Suggest

The suggest component is inspired by Google Suggest [Google2004], an AJAX web application which suggests possible query strings after each new character is entered into an input box. For example if the letters "*hp*" are typed Google Suggest might return "*hp printers*" and "*hp laptops*" as possible suggestions.

As with other AJAX applications, the suggest component consists of a client and server. The client-side used a modified version of a JavaScript AutoSuggest widget [Kepley2005]. Additional functionality was added to allow the suggestions to be looked up from an external web service. The same dynamic script tag method used for retrieving schemas was also used to retrieving suggestions.

On the server side a web service was written as a servlet for Apache Tomcat. It receives requests from the browser in the form:

```
/suggest.js?template=Infobox_Company&field=name&pattern=He
```

The servlet responds with an XML document containing a list of potential strings that the user could

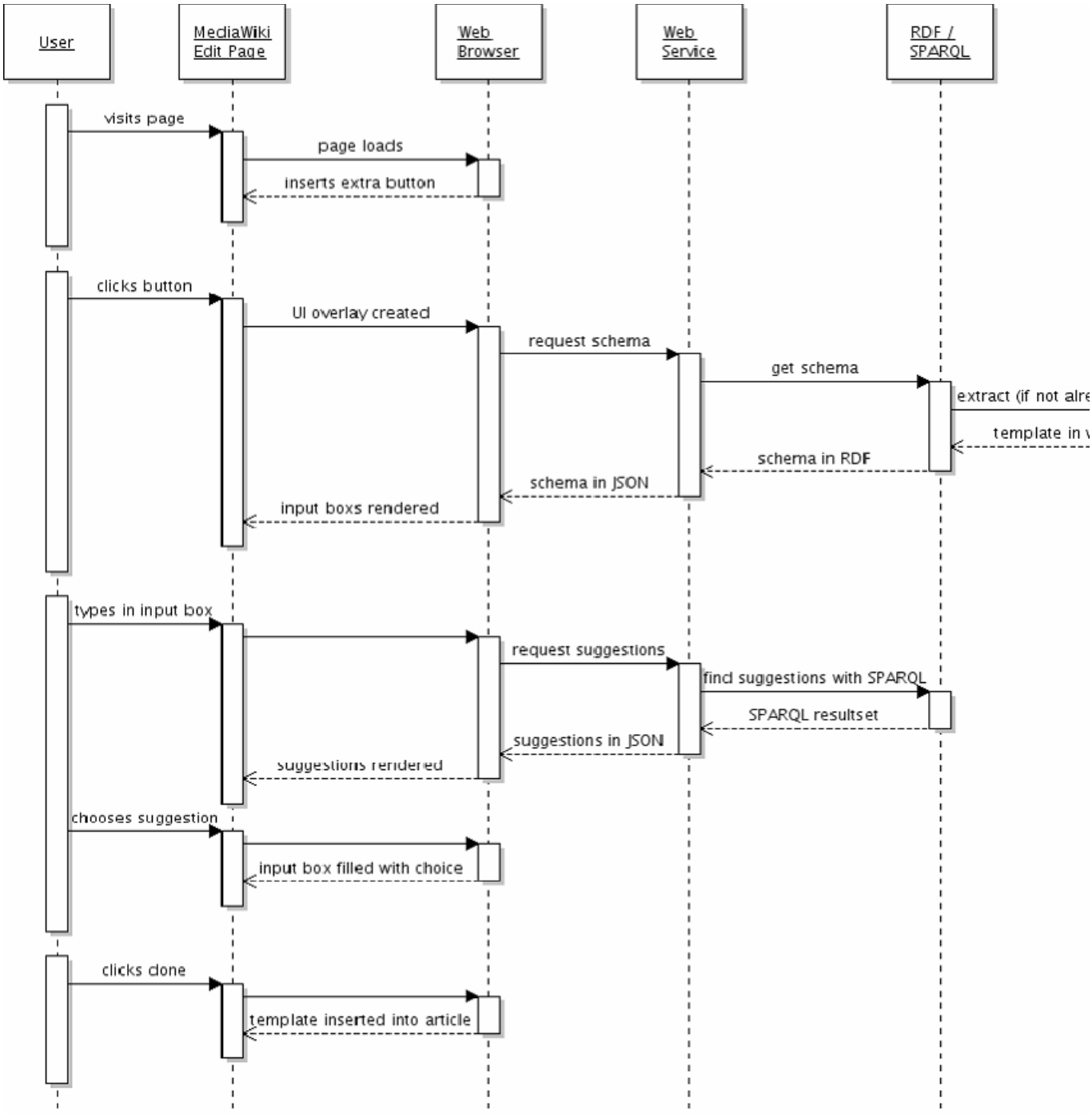

Figure 6 - UML sequence diagram showing interaction of components

choose to complete the field. Potential strings are chosen by querying the template instances which are cached in a local RDF store using Jena's implementation of SPARQL. SPARQL is a query language for RDF [Prud'hommeaux2007] that has some similarities to SQL.

The query is first restricted by the namespace of the template, then the field and finally the pattern that has been typed. This ordering is used to make the query more efficient as there will be fewer records to pattern match against.

Here is an example SPARQL query used for this task:

```
PREFIX a: <http://en.wikiped...Company>
SELECT DISTINCT ?o WHERE {
  ?s a:name ?o .
  FILTER regex(?o, "^H", "i") .
}
LIMIT 10
```

The current implementation has reasonable performance for small data sets, but some optimization will be necessary for larger data. For example it might be possible to make use of LARQ [LARQ], a version of ARQ which uses a Lucene index, in order to make the pattern matching faster. Another approach would have been to store the records in a String Trie [Fredkin1960], which is a tree where each node is a character from the key.

5 Phase Two: Conclusions and Further Work

The prototype developed in Phase Two demonstrates potential solutions to the three problems that we defined in the introduction:

1. **Ensuring re-usability of structured data**. We have demonstrated that a large amount of the structured data, including schema data, stored in Wikipedia in templates can be extracted into RDF. There are a few potential complexities such as the embedding of templates such as date of birth, determining data types of fields and the absence of a standard list format in WikiText.
2. **People find it difficult to remember technical details such as namespaces**. By extracting the WikiText template schemas and presenting the fields from the template as input boxes within a form, the technical details such as namespace URIs are hidden from the user.
3. **People naturally encode the same concept in different ways**. We have addressed this problem by implementing a "suggest" feature within the UI which suggests values for fields based on the field being completed and the characters entered.

5.1 Relationship to DBPedia

As already mentioned the DBPedia project [DBPedia], [Auer2007] has already done work on extracting structured data from Wikipedia as RDF. We did not use DBPedia in this work because we wanted to better understand the extraction process, but clearly the extraction process is difficult and the prototype we have developed here is less mature than the work currently being done by DBPedia. If this work is taken further, it would be worth revisiting the decision whether it is appropriate to use DBPedia. Interestingly though we made a number of alternative design decisions compared to DBPedia:

First, in DBPedia all properties are in the same namespace. Therefore if two pages use a property with the same name, it will be given the same URI. This assumption has been made for ease of query. Because of the number of authors on Wikipedia, and also the range of subjects, we did not wish to make this assumption, because it is problematic for properties like `type` which are used in a wide range of templates. Instead we consider other ways of assigning URIs to properties.

Second, DBPedia does not make use of redirect information. This is a deliberate design decision because the DBPedia decided that redirects contain information that certain terms are preferred over others. However, for naïve users this can be confusing: for example DBPedia does contain data about "*Hewlett-Packard*" but not "*Hewlett Packard*" whereas on Wikipedia this distinction is not apparent.

Third, at the time of writing, DBPedia does not extract schema information from Wikipedia. We think extracting schema information is potentially very useful.

Fourth, DBPedia does not, as far as we know, extract information from free text, only from templates and Pagelinks. In this work we explore this in addition to structured extraction techniques.

5.2 Further Work

There are a number of possible areas of future work arising from the work in Phase Two.

5.2.1 Better Presentation of Forms

Currently the form fields within the UI interface are rendered in the same order that they are retrieved

from the RDF model i.e. a random order. The UI would be more user-friendly if the input fields retained a consistent order. At present there is no agreed standard schema for describing presentation of a specific vocabulary in RDF, although there is a proposal for presentational RDF markup called Fresnel [Lee2007]. By using Fresnel, or a similar approach, it would be possible to encode ordering information in the schema, derived from the order which the fields occurred within the WikiText template schema.

5.2.2 Merging Namespaces

As discussed in Section 5.1.3, many templates use the same property. One way to avoid this problem is to assume all properties with the same name have the same URI, as done in DBPedia.

However, let us assume we adopt the naming approach used here, i.e. that we should not assume that two properties used in different templates are the same because they have the same name. Then in addition to using WikiLinks to link properties to definitions, we would like to propose two other ways of explicitly reusing properties in different templates in Wikipedia:

1. In the template definition, add some syntax so it is possible to map the field name to field name in another template.
2. Support the stacking of multiple template instances within a single article.

The first alternative involves mapping fields to fields defined in other templates within the template definition. This could be done as follows:

```
{{{field->otherTemplate_fieldName}}}
```

One advantage of doing this in the template definition is this is done by template editors rather than Wiki contributors. This is advantageous because understanding mappings between templates may be difficult for contributors.

In the second alternative, a number of Infoboxes are stacked on a page. For example a page about a musical artist might use a general person template and then a more specific template which contains fields relevant to music artists:

```
{{Person
| firstname = blah
| lastname = blah
}}
{{Artist
| aliases = sumer cool
| instrument = piano
}}
```

Splitting up Infoboxes in this way reduces the number of templates that contain a specific property, avoiding the problem that multiple templates use the same property. However we note that [Auer2007] argues against having multiple templates on the same page but does not give an explanation why.

5.2.3 Determining data types

As discussed in Section 6.1.4 there is some difficulty in determining the data types of fields. However by examining a number of instances of a template, some but not all will contain information that can help determine the type. Consider this example:

```
revenue = {{profit}} [[United States
dollar|US $]]91.7 [[1000000000
(number)|billion]] (2007)
```

Here we could have deduced that this property is talking about currency and money, as United States Dollar is a type of currency. Future work could investigate the implementation of an algorithm which several instances of fields of the same template and knowledge of common formatting to automatically classify a field to be of a certain type, which can then be encoded in the Schema using XML Datatypes [XSD2004].

6 Conclusions

In conclusion, we have described how Wikis can be used to create structured information in the form of RDF. In Phase One we created a prototype that demonstrated this by providing a novel browse view on information about companies and acquisitions. Then in Phase Two we investigated more generic extraction of data and schemas. We also reused this information to create forms that support AutoSuggest to help users create more structured data.

In the process we highlighted a number of difficulties with performing these tasks on Wikipedia, which stem from the fact that the structured data was created to be rendered using a stylesheet rather than reused

programmatically. However we are optimistic that demonstrating how this structured data can be reused and browsed in novel ways or support authors provides a compelling argument for this type of data and the creation of tools that simplify authoring such data.

7 Bibliography

[Auer2007] S. Auer and J. Lehmann, "*What have Innsbruck and Leipzig in common? Extracting Semantics from Wiki Content*", The Semantic Web: Research and Applications, pages 503-517, 2007, http://www.eswc2007.org/pdf/eswc07-auer.pdf

[Bizer2007] C. Bizer, R. Cyganiak and T. Heath, "How to Publish Linked Data on the Web", 2007, http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/

[Britannica2006] "*Fatally Flawed*", Encyclopedia Britannica, March 2006, http://corporate.britannica.com/britannica_nature_response.pdf

[Cunningham2002] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, "*GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications*", Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02), Philadelphia, July 2002, http://gate.ac.uk/

[Cunningham2005] H. Cunningham, "I*nformation extraction, Automatic*", Encyclopedia of Language and Linguistics, 2nd Edition, 2005, Elsevier, http://gate.ac.uk/sale/ell2/ie/main.pdf

[DBPedia] "*DBPedia*", http://dbpedia.org/

[Doctorow2001] C. Doctorow, "*Metacrap: Putting the Torch to the Seven Straw-Men of the Meta-utopia*", 2001, http://www.well.com/~doctorow/metacrap.htm

[Economist] "*Sharing what matters: A computing maverick hopes to upgrade the web, transforming it from a document collection into a data commons*", The Economist, June 7th 2007, http://www.economist.com/printedition/displaystory.cfm?story_id=9249171

[Fredkin1960] E. Fredkin, "*Trie Memory*", September 1960, Communications of the ACM, 3(9):490-499

[Garrett2005] J. J. Garrett, "*Ajax: A New Approach to Web Applications*", 2005, http://www.adaptivepath.com/ideas/essays/archives/000385.php

[Giles2005] J. Giles, "*Internet encyclopaedias go head to head*", Nature, Volume 438, Number 7070, 14 December 2005, pages 900-901, http://www.nature.com/nature/journal/v438/n7070/full/438900a.html

[Good2002] J. Good, "*A Gentle Introduction to Metadata*", 2002, University of California, Berkeley, http://linguistics.berkeley.edu/~jcgood/bifocal/GentleMetadata.html

[Google2004] Google, "*Google Suggest FAQ*", 2004 http://labs.google.com/suggestfaq.html

[Greasemonkey] "*Greasemonkey*", http://www.greasespot.net/

[Grimes 2005] S. Grimes, "*Is 'Unstructured' data merely unmodeled?*", Intelligent Enterprise, 1 March 2005, http://www.intelligententerprise.com/showArticle.jhtml?articleID=59301538

[Grisham1997] R. Grisham, "*Information Extraction: Techniques and Challenges*", SCIE, 1997, pages 10-27, http://citeseer.ist.psu.edu/grishman97information.html

[Herrington2006] J. D. Herrington "*The Ajax transport method: There's more to Ajax than XMLHttp*", June 6 2006, IBM developerWorks, https://www6.software.ibm.com/developerworks/education/x-ajaxtrans/x-ajaxtrans-ltr.pdf

[Huynh2007a] D. Huynh, "*Exhibit*", MIT, 2007, http://simile.mit.edu/exhibit/

[Huynh2007b] D. Huynh, "*Exhibit For Authors*", MIT, 2007, http://simile.mit.edu/wiki/Exhibit/For_Authors

[Jacobs2004] I. Jacobs and N. Walsh, "*Architecture of the World Wide Web: Volume One*", W3C

Recommendation 15 December 2004, http://www.w3.org/TR/webarch

[JavaRegex2007] "*Java Class Pattern: A compiled representation of a regular expression*", *Java™ 2 Platform* Standard Edition 5.0, Sun Microsystems, http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html

[JDIL] "*Java Data Integration in JSON*", http://jdil.org/

[Jena] "*Jena Semantic Web Framework*", http://jena.sourceforge.net/

[JSON] "*Javascript Object Notation*", http://www.json.org/

[Kepley2005] J. Kepley, "*AutoSuggest Sample*", http://gadgetopia.com/autosuggest/

[Kuhlins2002] S. Kuhlins and R. Tredwell, "*Toolkits for Generating Wrappers*", Lectue Notes in Computer Science, Volume 2591, NetObjectDays International Conference, Pages 184-198, 2002, http://www.old.netobjectdays.org/pdf/02/papers/node/0188.pdf

[Lee2007] R. Lee, "*Fresnel – Display Vocabulary for RDF*", World Wide Web Consortium, http://www.w3.org/2005/04/fresnel-info/

[LARQ] "*LARQ (Lucene + ARQ) Free Text Indexing for SPARQL*", http://jena.sourceforge.net/ARQ/lucene-arq.html

[Manning2005] C. Manning, "*Web Search and Mining: Lecture 3, Information Extraction*", 2005, Stanford University, http://www.stanford.edu/class/cs276b/syllabus.html

[Manola2004] F. Manola and E. Miller, "*RDF Primer*", W3C Recommendation 10 February 2004, http://www.w3.org/TR/rdf-primer/

[McGuiness2004] D. McGuinness, F. van Harmelen, "*OWL Web Ontology Language Overview*", W3C Recommendation 10 February 2004, http://www.w3.org/TR/owl-features/

[MediaWiki] "*MediaWiki*", http://www.mediawiki.org/

[O'Reilly2007] T. O'Reilly, "*Freebase will prove addictive*", O'Reilly Radar, 8 March 2007, http://radar.oreilly.com/archives/2007/03/freebase_will_p_1.html

[Prud'hommeaux2007] E. Prud'hommeaux and A. Seaborne, "*SPARQL Query Language for RDF*", World Wide Web Consortium, 2007, http://www.w3.org/TR/rdf-sparql-query/

[Soderland1999] S. Soderland, "*Learning Information Extraction Rules for Semi-structured and Free Text*", Machine Learning, Volume 34, Number 1-3, pages 233-272, 1999, http://citeseer.ist.psu.edu/soderland99learning.html

[Sun2007] "*Java Reflection*", Sun Microsystems, 2007 http://java.sun.com/javase/6/docs/technotes/guides/reflection/

[Tablan2003] V. Tablan, "*CS3421 Natural Language Engineering: Information Extraction*", 2003, University of Manchester, http://www.cs.man.ac.uk/~mary/CS3421lectures/node19.html

[Turmo2006] J. Turmo, A. Ageno, N. Catala, "*Adaptive Information Extraction*", ACM Computing Surveys, Volume 38, Number 2, Article 4, 2006, http://doi.acm.org/10.1145/1132956.1132957/

[WikiNamespaces] "*Wikipedia Namespaces*", http://en.wikipedia.org/wiki/Wikipedia:Namespace

[Wikipedia2007] "*Wikipedia*", 2007, http://en.wikipedia.org/wiki/Main_Page

[XSD2004] P. V. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium, October 2004, http://www.w3.org/TR/xmlschema-2/