

Draft Dissertation:
Constructing and Populating Ontologies from
Text Documents and Database Fields

Kate Byrne



Doctor of Philosophy
Institute for Communicating and Collaborative Systems
School of Informatics
University of Edinburgh
2007

Abstract

Hybrid databases, consisting of structured relational tables associated with free text documents, are a common way of holding information. This work concerns itself with the cultural heritage domain, where such databases are the norm. There are three significant problems facing the non-expert user trying to run effective queries against these databases: knowledge of the individual relational database structure is needed, specialist terminology must be used, and the text documents cannot usually be searched at the same time as the relational tables. For the terminology issue, there are a large number of cultural heritage thesauri available, but arguably they are too numerous and too complex to be truly helpful to the non-expert. Building a definitive ontology for any active domain of knowledge, where ideas are constantly evolving, may in any case be an impossible task.

The first claim is that the semantic content of the text documents can be adequately captured as a set of binary relations forming a directed graph. The ancillary data can be transformed into the same structure, which will therefore comprise relations derived from published domain thesauri or ontologies, relational database fields transformed to binary relations and the relations extracted from text. The set of predicates (or graph edge labels) is hand-constructed, to be as small as possible and to be generic across cultural data. Unlike a formal ontology, the graph will not necessarily be consistent across its entire extent. For convenient reference, this structure will be called a *datagraph* here.

The next hypothesis is that querying against the datagraph will remove the three query problems listed above. The structure is uniform and very simple, and the datagraph can be queried even if the predicates are unknown. The specialist terminology is integrated in the datagraph, which will enable query expansion or reformulation. The text content is also included.

Finally, it is claimed that use of the datagraph will produce additional benefits for querying, partly through natural graph characteristics and partly through the structure's particular design. The length of the path between pairs of nodes will be taken to indicate how closely relevant they are to each other, so preferring shorter path lengths can be used to resolve data inconsistencies (whereas in a relational database there is no equivalent concept of locality). A mechanism is described for summarising intermediate query results by grouping the attributes of nodes with shared predicates, which will place the user's query in a meaningful context. This is a consequence of the design of the predicate set; in a generalised graph the computational complexity would preclude

practical results. Thirdly, it may be possible to demonstrate the uncovering of connections between neighbouring entities that were hidden previously: in the datagraph mere proximity is sufficient to indicate relatedness, whereas in the relational database any connection between data items can only be extracted by an explicit query, i.e. by knowing in advance that the connection is there.

These claims will be tested by comparing queries across the original and the new data structures. The datagraph must be able to answer correctly queries that the original database dealt with, and must also demonstrate valid answers to queries that could not previously be answered or where the results were incomplete.

Table of Contents

1	Introduction	1
2	Central Ideas	3
3	Related Work	5
4	The Data	7
4.1	Core Data Collections	7
4.2	Domain Thesauri	9
4.3	Annotation	11
4.3.1	Named Entity layer	11
4.3.2	Relations Layer	17
5	Converting RDBMS Data to RDF	23
5.1	Background	24
5.2	Mapping to URI Space	24
5.3	Dealing with Relational Joins	26
5.4	Schema Design	29
5.5	RDF Bulk Load Performance	31
5.6	Other Conversion Issues	32
5.7	Implications for Graph Querying	33
5.8	Summary	34
6	Named Entity Recognition	37
6.1	Nested Entities	37
6.2	Experimental Setup	39
6.3	Results	42
6.4	Discussion	48

6.5	NER Across the Entire Dataset	49
7	Relation Extraction	51
8	Graph Queries	53
8.1	RDF Compared with RDBMS	53
8.1.1	Experiments	54
8.1.2	Results	55
8.2	Extending the Graph with Relations from Text	55
8.2.1	Experiments	55
8.2.2	Results	56
9	Extending to Related Domains	57
10	Conclusions	59
A	Tethering Cultural Data with RDF	61
B	Timetable for remaining work	73
	Bibliography	75

List of Figures

4.1	Graph generated from Monument Type terms	10
4.2	Example of relations from text	12
4.3	Translation of relation tuple to RDF	21
4.4	CREATION event with multiple agents	22
5.1	Efficient translation of RDBMS tables to RDF	28
A.1	Translation of RDBMS tables to RDF	65
A.2	Representation of graph schema	66
A.3	Graph generated for terms “cairn” and “chambered cairn”.	67

List of Tables

5.1	Conversion of RCAHMS data to triples	31
6.1	NE types distribution	40
6.2	Summary of NER results	43
6.3	Detailed NER results	46
6.4	Examples of NER errors	47
6.5	“Unlabelled” results for Table 6.3 runs	48
6.6	Comparison using ZLMaxent	48

Chapter 1

Introduction

A short chapter, not yet written. It will cover:

- the motivation for the work — improving access to cultural data by non-experts
- the methodology, in broad terms — convert everything to triples
- the academic contribution — translating relational database joins to RDF, relation extraction work, graph querying across hybrid data
- an outline of the structure of the rest of the document.

Chapter 2

Central Ideas

This chapter will explain the main components of my programme of work. Briefly, these are:

1. Justification of the graph database format.
2. Transforming structured data, free text and domain thesauri into a common graph structure.
3. Some explanation of the transformation of free text: NER and relation extraction.
4. The ontology structure: semi-automatic construction, followed by automatic population.
5. Querying the graph.
6. Possible future extensions to the work.

Some of this material will come from a previous paper Byrne (2006), which is at Appendix A for reference.

Chapter 3

Related Work

This chapter will include material from the Thesis Proposal's literature review (section 2) and from the relevant sections of my formal papers. It will need to be updated with more recent material.

Chapter 4

The Data

This chapter gives an overview of the source material being used. Although the techniques being used are intended to be as generic as possible, the motivation for this work is closely tied to cultural heritage data management issues, and there are features specific to data from this domain. Section 4.1 considers the composition of the core data obtained from several heritage bodies, and its particular characteristics.

The cultural domain has no shortage of specialist thesauri available, and material from some of the standard sources is incorporated in the constructed ontology. Section 4.2 describes this information and the way it is transformed to be used alongside the core data.

A randomly chosen set of documents derived from the RCAHMS dataset was annotated to produce a gold standard set for evaluating the Named Entity Recognition and Relation Extraction tasks. Smaller extracts from the other two core datasets were annotated to the same rules — which are explained in Section 4.3 — to be used in cross domain testing.

4.1 Core Data Collections

The theme of this of this programme of work is to assist non-expert users in querying heritage databases and their associated free text documents. Three of the Scottish National Collections have contributed data for the project:

- **RCAHMS**¹ — a snapshot of the National Monuments Record of Scotland (NMRS), containing 250,000 records and texts about archaeological sites and historic

¹The Royal Commission on the Ancient and Historical Monuments of Scotland, <http://www.rcahms.gov.uk/>.

buildings, and 750,000 associated archive item records (photographs, maps, plans etc.). This data covers two domains: archaeology and architecture.

- **NMS**² — 114,000 archaeology database records with associated texts, describing Scottish excavation finds and other archaeological objects.
- **NLS**³ — 20,000 records in **SCRAN**⁴ format, describing items from the NLS historical collections including books, broadsheets, photographs and maps. The significance of the SCRAN format (fixed fields and free texts, like the other databases) is that almost every cultural organisation in Scotland has repositories of data in this layout, having contributed either to SCRAN or to its sister project RLS.⁵

These three datasets — or four, if one takes the RCAHMS archaeology and architecture separately — cover different but closely related domains, and have a lot of specialist terminology in common. The RCAHMS and NMS sets are the closest in terms of language and subject: the RCAHMS archaeology set describes sites in Scotland and the NMS set covers finds from such sites. There is no existing formal correlation between the sites covered by each dataset, but it is certainly the case that many historical sites feature in both sets. The NLS data is rather different in character, and part of the experimental work will involve ascertaining whether a machine learning model trained on the RCAHMS entity and relation framework will be successful in the NLS and SCRAN domain. The end goal is to see if the techniques developed for one (RCAHMS) can be extended to the others, and if useful querying can be done across multiple datasets.

The organisations are keen to extend access to their collections to a much wider public than in the past, and have done market research to assess the needs of their audience (Turnbull, 2003) and evaluate their existing web-based provision (Bailey, 2004). (Of course, there is also non-web access, but it is not our concern here.) These surveys show that the aim should be to reach the general public and users at almost all levels of education, from school curriculum through higher education to “lifelong learning”. Since most of the material has been developed, over many decades, for specialists and professionals, this is a very tall order.

It may be a less pressing priority, but there is also a desire to make the material available to non-English speakers by translating it into other languages. Clearly this

²National Museums of Scotland, <http://www.nms.ac.uk/>.

³The National Library of Scotland, <http://www.nls.uk/>.

⁴Scottish Cultural Resources Access Network, <http://www.scran.ac.uk/>.

⁵Resources for Learning in Scotland, <http://www.rls.org.uk/>.

would be extraordinarily time-consuming and expensive to do manually. At present some of the organisations mentioned above are looking at the possibility of translating just the query interface, to allow searching in another language but with results in English.

4.2 Domain Thesauri

There are many, many specialised thesauri for the cultural heritage domain with which I am dealing.⁶ Most of them are freely available, and all are used to some extent by the three organisations whose data I am working with. Crofts et al. (2003) provide an overview of an attempt by CIDOC (Comité International pour la Documentation) to harmonise efforts, under the *Conceptual Reference Model*.

A representative set of these standard thesauri will be translated into RDF triples in the same way that the fixed database fields are translated (see Chapter 5). The W3C SKOS⁷ (Simple Knowledge Organisation System), which is designed for handling thesauri, may provide the implementation framework. Figure 4.1 shows an example of how such an RDF graph might appear. The subset shown is the context of the two terms “cairn” and “chambered cairn”, from TMT (Thesaurus of Monument Types).

The standard thesauri comprise three core relation types: *Hierarchical*, *Associative* and *Equivalence* relationships (MIDAS, 2003; Tudhope and Binding, 2004). As is explained in Section 4.3, the set of relation types used in the annotated corpus includes these three, labelled respectively *isA*, *seeAlso* and *sameAs*. Thus, the extraction of relations can be viewed as an ontology building exercise. The ontology structure derived from domain thesauri will be augmented with relations extracted from text documents. It may also be possible to use the standard thesauri to validate extracted relations, rejecting ones which would make the graph inconsistent.

⁶To name just a few: TMT (Thesaurus of Monument Types, maintained by English Heritage), FISH (Forum on Information Standards in Heritage, for the UK and Ireland, maintained by the MDA (which formerly stood for “Museum Documentation Association”)), SPECTRUM (UK Documentation Standard for museums, maintained by MDA), AAT (Art and Architecture Thesaurus, one of the Getty vocabularies), ULAN (Union List of Artist Names, from the Getty), TGN (Thesaurus of Geographic Names, the third of the Getty set), TGM (Thesaurus for Graphic Materials, for image indexing, from the Library of Congress) LCSH (Library of Congress Subject Headings, much used in the library world).

⁷<http://www.w3.org/2004/02/skos/>

4.3 Annotation

A collection of 1546 documents from the RCAHMS dataset was annotated with Named Entities (NEs) and relations. This supplements an earlier annotation layer created for the SEER project in 2003; see Nissim and Krymolowski (2003) for a description of that. This section describes the new entity and relations annotation, which is what was used in all the experiments.

The document files are a randomly chosen subset of the text notes fields from the RCAHMS database; one text field per file. They vary in length from a couple of lines to about a page. The text is in the form of rough notes: only a minority of the snippets form grammatical English sentences.

4.3.1 Named Entity layer

The entity classes are: ORG, PERSNAME, ROLE, SITETYPE, ARTEFACT, PLACE, SITENAME, ADDRESS, PERIOD, DATE, EVENT. The SITETYPE, EVENT and ARTEFACT classes are further divided into subclasses, as described in Section 4.3.1.1 below. Co-reference is treated as a relation between entities, as explained in Section 4.3.2. This is in contrast to the SEER annotation, which used a LINK class. Two other classes from the SEER annotation were dropped: SIZE and REFERENCE, the latter because bibliographic reference is treated as a DESCRIPTION event, with an associated eventRel relation (see Sections 4.3.1.12 and 4.3.2.7). Entities may be nested, with a maximum of three levels — as in [[[Edinburgh]^{PLACE} University]^{ORG} Library]^{ORG}, for example.

4.3.1.1 Instances and Classes

Each text string marked as a Named Entity is automatically given a unique identifier by the annotation software. For all entity classes except SITETYPE and EVENT this is all that is required, as the members are distinct individual instances, such as “Mr. J.D. Jamieson” (a member of the set of PERSNAMEs) or “15 May 1968” (a DATE).

The members of SITETYPE are strings like “chambered cairn”, which may be generic or specific: compare “the Dwarfie Stane is a chambered cairn” (generic) with “the chambered cairn is in Hoy And Graemsay” (specific). For the specific case, the unique instance identifier given by the software is used in any relation marking. In the generic case, “chambered cairn” is the name of a set — of entities that are chambered

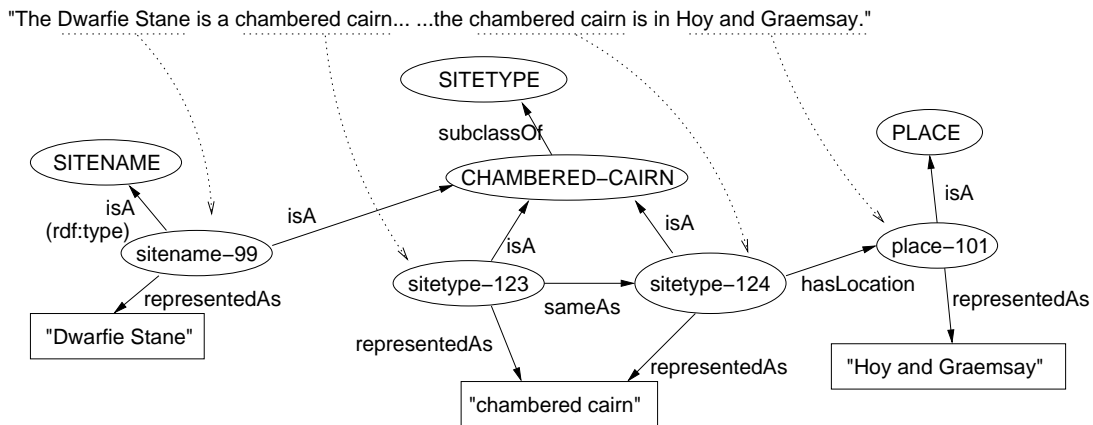


Figure 4.2: Example of relations from text, showing introduction of CHAMBERED-CAIRN class as subclass of SITETYPE.

cairns. Therefore an extra class such as CHAMBERED-CAIRN is needed, which is a subclass of SITETYPE, and of which the specific instance (say sitetype-123) is a member. Figure 4.2 illustrates the point. To think of it in terms of relational algebra, the strings associated with SITETYPES sometimes represent variables and sometimes values. In contrast, PERSNAMEs (for instance) are always values, of type PERSNAME.

The implication is that entities classified as SITETYPE must have a subclass identified. These are taken from the Thesaurus of Monument Types, which includes several hundred terms.

Much the same applies to EVENT entities, but the set of subclasses is more manageable: SURVEY, EXCAVATION, FIND, VISIT, DESCRIPTION, CREATION and ALTERATION. Each text string marked as an EVENT gets an identifier (say event-456) and is a member of one of the EVENT subclasses, say VISIT. See Section 4.3.1.12 for further details.

ARTEFACT entities are also sub-categorised, as described in Section 4.3.1.6 below. However, these NEs do not in general act as the range for relations; unlike SITETYPE entities, they are almost exclusively used as instance terms not class terms. In this case the MDA Object Type Thesaurus is the source for the subclass labels.

4.3.1.2 ORG

The ORG class covers organisation names such as “RCAHMS”, “Ordnance Survey”, “OS” etc. It is also used for named archive collections (usually identified in the text by the word “Collection” following the entity string) or architectural practices, such as

“Walker & Duncan”. In cases like this the component entities will also be tagged (as nested entities), typically as PERSNAME.

4.3.1.3 PERSNAME

The PERSNAME class covers personal names and initials representing individuals, such as “Vere Gordon Childe”, “Henshall”, “RJB” etc.

4.3.1.4 ROLE

This class is for the rôles of agents (usually PERSNAMEs), where these are specifically mentioned, for example “architect”, “donor”, “Chief Executive”. In the archaeological data it occurs only rarely.

4.3.1.5 SITETYPE

The SITETYPE class includes site classification terms like “chambered cairn”, “long barrow”, “cist” and so forth. These are marked as SITETYPE entities and assigned to a subclass selected from TMT (the Thesaurus of Monument Types), as explained in Section 4.3.1.1. Where the closest matching term is not obvious it is a matter of the annotator’s judgment to pick the best TMT entry to serve as the subclass name.

4.3.1.6 ARTEFACT

This label covers terms denoting physical objects that have a significant relationship with the parent site described by the document, yet are distinct from it rather than an integral part. This includes archaeological finds such as “bronze axe”, “sword”, “pottery shards” and so on. Despite the name, it is not restricted to man-made objects, but includes items such as “human remains”, “dog bones”, “seeds”, etc. It is not used for objects that are components of the parent site, such as “the second sheiling” or “this mound”. As a general rule ARTEFACTs are portable items that could in principle be separated from the parent site without losing their identity. Each ARTEFACT entity is given a class label from the Object Type Thesaurus, in the same way that SITETYPES are subclassified.

There are cases where the choice between SITETYPE and ARTEFACT is somewhat arbitrary — where there is a description of inhumations (human burials) for example. If the parent site is a burial site, and the entity term refers to, say, a burial chamber which is part of the site, SITETYPE is used. Where the term refers to the

remains contained in a burial then ARTEFACT is used. There are cases where the annotator has had to make a best guess.

4.3.1.7 PLACE

There are three locational entity classes: PLACE, SITENAME and ADDRESS. PLACE is used for all administrative place names, such as regions, districts, parishes, counties and countries. It includes the kind of names that might appear on an Ordnance Survey map, such as “Dumfries and Galloway”, “River North Esk”, “Arthur’s Seat”.

4.3.1.8 SITENAME

The SITENAME class is used for terms that name a specific site, such as “Skara Brae”, “Maes Howe”, “Stones of Stenness”.

4.3.1.9 ADDRESS

This class is intended to pick up terms that describe the location of a site, rather than simply naming it. Part of the aim was to avoid cluttering the PLACE and SITENAME classes with very local terms that are unlikely to be useful for querying the final graph. For example, architecture texts often include entity terms such as street names, or even postal addresses. In archaeological texts grid references are common, such as “HU 3754 3380” and site references like “HU33SE 43”,⁸ and both will be marked as ADDRESS. This label is also used for private places like “John’s farm” (unless they indicate a specific archaeological site, when SITENAME is used). Names that are too local for the “map rule” applied to the PLACE class — such as street names or house names — are classified as ADDRESS.

Inevitably, there are cases where the distinction between PLACE, ADDRESS and SITENAME is somewhat arbitrary. For example, “Hill of Caldback” is a SITENAME because it happens to be an archaeological site, but somewhere with no particular historical significance, like “Blackford Hill”, would be a PLACE. A location like “Braes of Doune” would be a PLACE, but “Liberton Brae” would be an ADDRESS in most contexts (as it is the name of a road in the city of Edinburgh).

⁸“HU33SE” is the number of a particular OS map sheet, and “43” is the number allocated by the NMRS for a particular site on that sheet. The combination constitutes a unique identifier for the site. There is often a sub-number as well, expressed as “43.1” or “43-1” or similar.

4.3.1.10 PERIOD

The PERIOD class covers terms such as “late Neolithic”, “16th Century”, “modern” and so on. Terms identifying calendar dates come under DATE.

Surprisingly, given the domain, PERIOD terms are rare in the documents and this class is a small one. The reason has to do with the unwillingness of professionals to commit authoritatively to a particular period when this may be a matter of dispute, combined with a reluctance to patronise expert readers — for whom most of the texts were originally written — by explicitly mentioning the period when it might be considered obvious. This is an example of where NLP techniques may be particularly helpful to non-expert users, in gleaning every possible reference to a fundamentally important piece of information that is largely absent from the searchable database fields. Speaking as such a non-expert user myself, I experienced a Gestalt shift when I realised that text snippets including “House 1; interior, detail of niche above bed. ... Interior of house 7. Hearth and dresser.” were referring to a Neolithic site rather than a relatively modern residential building.

4.3.1.11 DATE

This class covers date values, such as “1st Jan 1980”, “October 2002”, “1454”, etc. Non-specific time references like “Sunday morning” or “the following week” are not labelled, as they are not plausible query terms. For the same reason, time expressions such as “11am” or “noon” are not included.

As with the locational classes above, but of less significance because these sets are much smaller, there is overlap between DATE and PERIOD. The former is primarily for calendar dates, but may include expressions like “1870–1880”; whilst PERIOD would include terms like “late 19th Century”. It is a moot point which class fits “the 1870s” better, and the decision would be made by the annotator in context. However, specific individual calendar dates are comparatively easy to recognise and these classes could easily be further manipulated in a post-processing step if it proved useful at the graph querying stage.

4.3.1.12 EVENT classes

This family of classes covers terms describing events in the history of a site, such as visits, surveys and excavations. The subclasses of EVENT are: SURVEY, EXCAVATION, FIND, VISIT, DESCRIPTION, CREATION and ALTERATION. The first five

event types listed are the ones most frequently encountered in this corpus.

The EVENT classes all pertain to n -ary relations that are awkward to translate into simple two-place predicates. Each event typically has an object or “patient” (generally the site itself), an agent (the instigator of the event) a date, and possibly other attributes or semantic roles (such as what was found, in a FIND event). These linked NEs are picked up through the relation annotation, as described in Section 4.3.2 below. The event entities are needed to provide a hook to hang the relation annotation on.

In many cases the events are implied by the text, not explicitly mentioned. For example, phrases like “visited by OS, 1990” are common; clearly a visit took place, but there is no noun phrase to label. In such cases the verb phrase (“visited”, in this case) is marked as a VISIT event entity. In other cases a suitable nominal is present, as in “...excavation carried out by...”, where “excavation” is an EXCAVATION event.

In more detail, the different subclasses cover events as follows:

1. *SURVEY*: This is the appropriate category when the text contains references to “plan”, “survey”, “measured survey”, “GDM survey”, “photographic record” or such like. It implies a detailed examination of the site resulting in the production of archive material.
2. *EXCAVATION*: Self-explanatory - there will be a reference to an excavation or dig, i.e. an event in which the site was deliberately physically disturbed by an archaeologist.
3. *FIND*: When an ARTEFACT entity occurs, there is typically an associated FIND-EVENT, which will generally be expressed as a verb phrase such as “was discovered”, “found” etc.
4. *VISIT*: This is a fairly unspecific event, when an organisation or person went to the site but there is no reference to a survey or excavation.
5. *DESCRIPTION*: This is the most general category, used when it’s not clear that the site was visited at all, but some agent is mentioned as having produced a tangible description or depiction. It is also used for bibliographic references, as follows. Where there is a suitable noun or verb phrase (as in “...described by E Beveridge”) this is marked as the DESCRIPTION entity (“described” in this case). Where there is only a free-standing bibliographic reference (such as “E Beveridge 1911”) the whole string is marked as a DESCRIPTION, with nested

entities inside it (typically PERSNAME or ORG, and DATE).⁹

6. *CREATION*: This category, and the next one, are uncommon in the RCAHMS archaeological data that comprises most of the corpus, but was included to provide coverage for architectural data. A *CREATION* event refers to the original construction of a monument. In the case of a building, several agents may be mentioned (architect, builder, patron, etc.) as well as a date. If there are multiple rôles or dates, separate *EVENTs* are used. (See Section 4.3.2.7 and Figure 4.4 for details.) As with *DESCRIPTION*, where there is no obvious noun or verb phrase (such as “built by”) available, a suitable string is identified for labelling, usually the compound of the entities participating in the event (e.g. “Architect: William Adam 1723”).
7. *ALTERATION*: This covers events that change the physical character of a site significantly, such as serious damage, extension, transfer of location, etc. Occasionally a monument that no longer exists is recorded, and there may be information on when it was destroyed. This is also classed as an alteration; the cases are too rare to warrant a separate *DESTRUCTION* class.

4.3.2 Relations Layer

The relation annotation covers the NE layer just described. With one exception, the relations are (subject, predicate, object) triples,¹⁰ where the subjects and objects are NEs. The predicates are: *isA*, *sameAs*, *seeAlso*, *partOf*, *hasLocation*, *hasPeriod*, *eventRel*. All except the last mentioned are triples as just described, but the *eventRel* relation has higher arity and is of the form: *eventRel*(*eventType*, *eventPatient*, *eventDate*, *eventAgent*, *eventRole*, *eventResult*). The *eventRel* relations are subsequently be transformed into binary relations, but the arrangement was intended to make the annotation process simpler.

The characteristics of the relations are as described below. Where possible relation names from published ontologies are used (such as *rdf:type*, *owl:sameAs*), but the renaming is done in a later processing step and is not detailed here. The most common domain and range of each relation is given but there are cases where they do not apply.

⁹The aim was to avoid nesting entities if possible, but some text string always needs to be marked as the NE that takes part in an *eventRel* relation.

¹⁰- or, equivalently, two-place relations of the form *predicate*(*subject*, *object*)

In linguistic terms the domain of a binary relation is typically the subject or agent in a textual expression, and the range is the object or patient.

4.3.2.1 isA

Domain: SITENAME
 Range: subclasses of SITETYPE
 Example: “Hill of Caldback is a chambered cairn”

This indicates membership of a class. Those isA relations which merely show the type or class of every single entity string, such as (“RCAHMS”, isA, ORG), is added automatically in a post-processing step when the extracted relations are converted to an RDF graph. In the annotation, the isA relation intended to indicate multiple inheritance, where an instance belongs to another class as well as its “natural” parent. In this domain, this is most likely to occur with membership of the SITETYPE subclasses, where the SITETYPE entity is used in a generic sense as was discussed in Section 4.3.1.1 and illustrated in Figure 4.2. Wherever possible the specific subclass is used (such as CHAMBERED-CAIRN) rather than its parent SITETYPE class. In the conversion to RDF, any entity that is the target of an isA relation will become a class label.

4.3.2.2 sameAs

Domain: any NE instance
 Range: NE instance of the same class
 Example: “...described by A. S. Henshall, 1985. Henshall also says...”

This is used for co-reference. In the original SEER annotation the LINK class was used, and only pronouns and definite descriptions were marked. For this new annotation layer, *all* co-referential mentions of entities are included, as the example illustrates. Any number of entities may be marked as belonging to the same co-reference set. They are typically entities likely to feature as nodes in the final graph, i.e. as the source or target of a binary relation. Hence relative pronouns, which were previously marked as LINKs, are not included.

Note that the relation applies to *instances* not classes. For the SITETYPE class, NE strings such as “chambered cairn” in “Hill of Caldback is a chambered cairn” will have a unique label assigned, such as sitetype-123, as has already been discussed (in Section 4.3.1.1). If the same entity is referred to elsewhere in the text as a “chambered

round cairn” (perhaps sitetype-456) the two SITETYPE instances will be linked by a sameAs relation. This does not imply that in general a chambered cairn is identical to a chambered round cairn, because those are class terms and the two classes are not equivalent.

4.3.2.3 seeAlso

Domain: any NE instance (often SITENAME or ADDRESS)
Range: often an NE instance of the same class
Example: “See also HU33SE 43 excavated by Parry”

This is intended for occasions when two entities are described as being closely related, or are contrasted with each other. It is also used when it seems sensible to link two entities, but the relationship between them is not in the specified set, such as a family tie (parent, child, sibling, etc.) between two PERSNAME entities, or some noteworthy association between, say, a SITENAME and a PERSNAME (such as “Sir Walter Scott lived here”). The relation is bi-directional, so the order of subject and object is not significant. However, for examples like the one given above, the convention followed is that the current site (the subject of the text) is the subject and the entity pointed at (in this case “HU33SE 43”) is the object.

The example shown illustrates a potential problem with the entity typing. The string “HU33SE 43” is clearly an ADDRESS by the rules defined above, yet in the context of this relation it should actually take the role of a SITENAME, in a peer-to-peer relation with the SITENAME that the document is about. Matters like this are not dealt with at this stage: the string is marked as ADDRESS, as makes most sense for the annotator and for the machine learner that will attempt to model the ADDRESS class. This issue, along with those surrounding generics, highlights the fact that NE typing is not always straightforward, and high precision and recall scores may not necessarily be very relevant. Inter-annotator agreement (IAA) figures will give an indication of how uncertain entity typing is for a particular corpus¹¹, but in cases like this one all annotators would presumably agree that “HU33SE 43” is an ADDRESS (given the rules provided), i.e. there would be a high level of agreement despite the class being wrong in this context. As it stands, this issue suggests a problem with the underlying thesis that the meaning of a collection of text documents can be adequately summarised as a set of relationships between entity terms within them.

¹¹IAA figures are needed for this corpus but have not yet been produced.

It may be that a rule-based post-processing step can iron out this kind of difficulty by changing the class of the “HU33SE 43” string. More interestingly, it may be possible to take advantage of inherent characteristics of the graph structure to produce what is actually the desired result: a connection between the two sites, discoverable by a query on either. This has not yet been explored, but the broad idea might be that particularly important entities (like those representing a SITENAME that is the subject of a document) will naturally gather a cluster of related nodes around themselves, and where `seeAlso` edges occur they should be treated as connecting the local cluster to some separate cluster. So there could be a process to search for the focus of the remote cluster, perhaps by looking for the node of highest degree within some pre-determined (small) path distance of the nodes at either end of a `seeAlso` edge.

4.3.2.4 `partOf`

Domain: SITETYPE
 Range: SITETYPE or SITENAME
 Example: “A farmstead comprising one unroofed building, two roofed buildings and one enclosure, and a head-dyke”

This is for part-whole relationships, such as when a complex site is described in terms of its components. In the example given, the “farmstead” SITETYPE is the whole, and “unroofed building” “roofed buildings”, “enclosure” and “head-dyke” are the parts. Each of them is a SITETYPE NE.

4.3.2.5 `hasLocation`

Domain: SITENAME, SITETYPE, ORG, PERSNAME, ARTEFACT
 Range: usually PLACE or ADDRESS; may be ORG (see below)
 Example: “...on the north side of the road leading to Bannaminn”

This relation covers cases where an entity is located at or in the vicinity of a PLACE or ADDRESS. In the case of rather vague descriptions like the one in the example, the consideration is whether knowing the location mentioned would help someone to find the entity. (In this case it would.) Negative examples (if they occur), such as “a long way from Edinburgh”, are therefore ignored. For ORGs and PERSNAMEs the relationship is typically with their addresses; for ARTEFACTs it is often with the museum holding them. If a PERSNAME is mentioned as belonging to an ORG, this is marked as `hasLocation`.

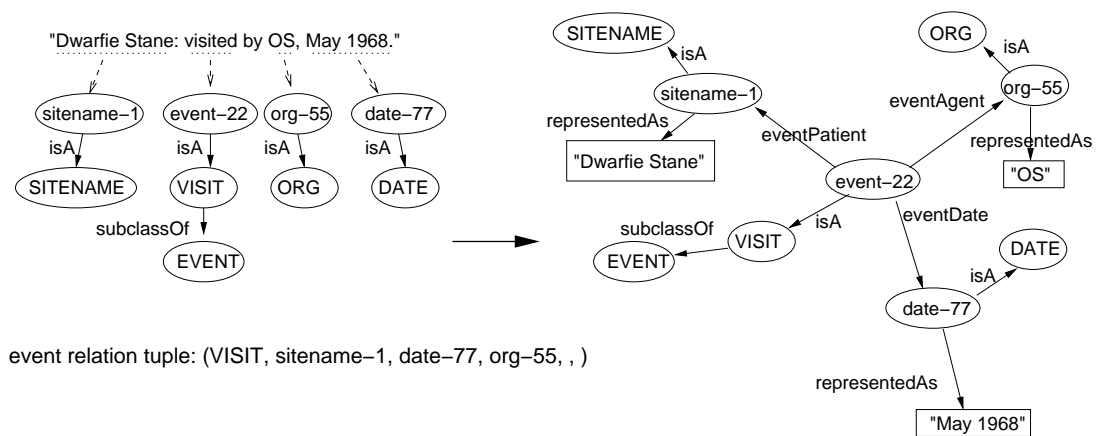


Figure 4.3: Example event relation tuple, and its translation to RDF graph format.

4.3.2.6 hasPeriod

Domain: SITENAME, ARTEFACT

Range: PERIOD

Example: “a late Viking potsherd”

This is a straightforward link to PERIOD NEs from the NE they apply to.

4.3.2.7 eventRel

Event relations connect a set of entities taking part in one of the events defined as subclasses of EVENT in Section 4.3.1.12. They will be converted to RDF binary relations in a later processing step. The tuple making up the relation is of the form: (eventType, eventPatient, eventDate, eventAgent, eventRole, eventResult).

Figure 4.3 shows the tuple for the example text “Dwarfie Stane: visited by OS, May 1968”, and how it can subsequently be translated into a graph of two-place relations.

In more detail, the arguments are:

1. eventType: one of the set {SURVEY, EXCAVATION, FIND, VISIT, DESCRIPTION, CREATION, ALTERATION}; required, and occurs once
2. eventPatient: the object undergoing the event, filling the direct object position for active verbs; null if no object mentioned, and can occur only once
3. eventDate: the DATE when the event took place; null if no date mentioned, and can occur only once.

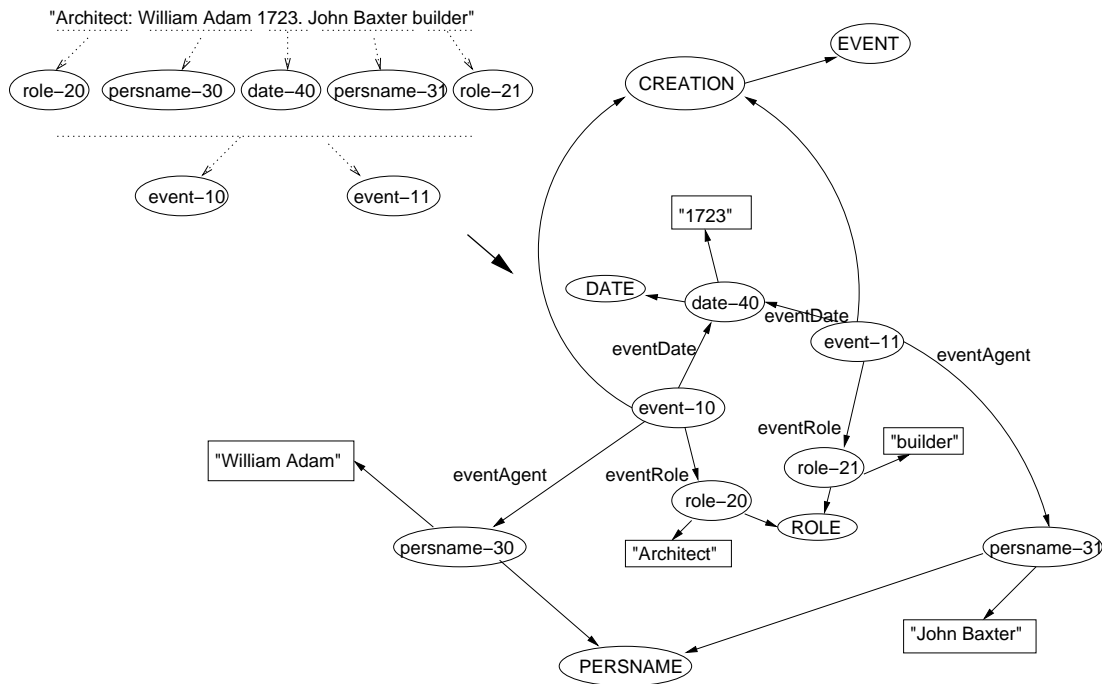


Figure 4.4: A CREATION event, split into two events related by sameAs, to cater for multiple agents and rôles. (Only the key edge labels are shown.)

4. **eventAgent**: the PERSNAME or ORG that was the instigator of the event; null if no agent mentioned, and can occur only once
5. **eventAgentRole**: the ROLE of the eventAgent, where this is explicitly mentioned (e.g. “architect”). In some events, such as CREATION, there may be multiple PERSNAME agents playing different rôles (architect, designer, etc.). In these cases, each eventAgent–eventAgentRole pair is put in a separate eventRel. See Figure 4.4 for an illustration. (The aim is to avoid the need for a more complex structure, with new relations, to tie each agent to the correct rôle.)
6. **eventPlace**: where the event took place, if this is obviously part of the relation; it is useful where some location *other* than the current SITENAME is mentioned; optional and non-repeating

Chapter 5

Converting RDBMS Data to RDF

Amongst the problems in providing internet access to cultural heritage databases — held in relational databases as they generally are — is that writing good SQL queries requires specialist knowledge of the often complex structure of the database in question. One of the theories being tested in this work is that conversion of this complex structure to a much simpler graph of RDF triples will make the query process easier. The goal is to create a query application (called *Tether*¹) that will guide non-expert users in constructing good queries, through summarisation and provision of relevant context. As a first step, the data is converted from its various relational database formats into a set of RDF graphs, and this chapter concentrates on the conversion process.

The RDF graph created from the RCAHMS dataset has over 15 million triples. This number is dramatically lower than the theoretical maximum required, for reasons discussed below. The options for mapping data items to URIs and for dealing with join conditions between relational tables are discussed, and the differences between relational and RDF schema design are examined. Some detail is given on particular problems encountered with a large, slightly untidy database, and performance issues are briefly covered. A few problems were encountered that had not been anticipated from previous experiments with much smaller quantities of data, and these are described. The data conversion exercise is the first step for a planned series of experiments comparing SQL querying over RDBMS tables with queries over RDF using SPARQL, which are described in Chapter 8.

¹“Tether” is a dialect word for “three”, used in the north of England for counting sheep.

5.1 Background

The material used for these experiments is a large subset of the RCAHMS dataset: around 250,000 site records with 750,000 associated archive items and 50,000 bibliography records. Held as a relational database — originally in Oracle, but transferred to MySQL for this project — it occupies about 380Mb including indexes. (This obviously excludes the terabytes of graphical material so far scanned to digital files.)

The basic process of translating a relational database to RDF triples is straightforward and easily automated. Every table or relation in the database can be turned into a set of triples in the form (*relationID*, *attribute*, *value*). Figure 5.1 illustrates the procedure. (Simple labels are shown in this diagram instead of URIs, for readability.) There is some compression of the relational joins in this method, discussed in Section 5.3 below.

The data was extracted from MySQL using SQL scripts, and loaded into the Jena RDF store,² configured to use a MySQL database for persistent back-end storage. The design of the store is described in HP (2004). Jena was chosen after a survey of the various RDF stores available, particularly those that implement persistent storage in a relational database back-end. See Cyganiak (2005), Harris (2005) and Alexander et al. (2005) for discussion of the issues involved in designing RDF storage.

5.2 Mapping to URI Space

One of the first things to be decided when implementing the translation procedure, is how to map RDBMS data items into RDF resources uniquely identified by URIs. As is discussed in Berners-Lee (2006), one reasonable option is to include the entire provenance of every data item every time it occurs: from the database name, schema and catalogue, down through the table name and attribute name, and including various other pieces of metadata such as the parent primary key and the SQL query that will extract the item. For those of us who remember when saving two characters in a date field was worthwhile,³ the data bloat is breath-taking. Given fixed field data where the data density is high (i.e. most fields are not null) and the average length of value strings is fairly short, the increase would probably be in the order of ten-fold, perhaps much more. Clearly storage space is not the issue it was in the past, but such an increase in

²<http://jena.sourceforge.net/>

³This practice led to the infamous “year 2000 problem”.

volume is still a drawback.

One aspect of this approach to identifying RDBMS data items is that it clearly implies that the RDBMS version is the master copy, to which the RDF points. In this case RDF translation is being used merely as a way of exposing a relational database on the Web. In due course one might expect data to be maintained in RDF — with the necessary graph updating languages this entails — in which case the earlier URIs would be superseded.

A compromise solution was adopted here. The URIs used refer to the source database name and include the data subject name or property name as appropriate, but do not include extra metadata. Although in the first instance the URIs were automatically generated from the RDBMS schema, they were later revised to group similar datatypes together, in order to facilitate planned future work involving summarising over sub-graphs. See Section 5.4 for discussion of the URI revisions.

The following are some examples of automatically generated statements⁴:

```
rc:site8864 rc:ngre    ``2902`` .
rc:site8864 rc:ngrn    ``7390`` .
rc:site8864 rc:mapno   ``ND27SE`` .
rc:site8864 rc:region  <http://ltg.ed.ac.uk/rcdb#regionHIGHLAND> .
rc:site8864 rc:district <http://ltg.ed.ac.uk/rcdb#districtCAITHNESS>
rc:site8864 rc:parish  <http://ltg.ed.ac.uk/rcdb#parishCANISBAY> .
```

These examples of RDF triples (expressed in N3 format) highlight the need to determine when the object node is a literal value and when a resource in its own right, with a URI. Clearly database key fields (primary or foreign) must be URIs, as they can be subjects of statements as well as objects of them. Equally, strings such as grid reference values are easily categorised as literals. But there are many data items where the choice is not immediately obvious. In the above examples the place names are given URIs, as they can form relationships with one another (Caithness is part of Highland region, for instance). They also have a conceptually generic quality that literal values do not. This classification of data values into literals or resources was done by hand, requiring knowledge of the database. At times it was rather arbitrary, as it seems difficult to pin down the distinction precisely.

The decision to concatenate an identifying string with key resource values (as in

⁴`rc:` is a URI prefix identifying RCAHMS as the data source, on a particular host machine: `<http://ltg.ed.ac.uk/rcdb#>`. It is not a valid URL, as serving the metadata on a website will postdate transferring it to RDF, if it happens at all. This is surely what would generally be the case.

site8864 and parishCANISBAY) was made because initial experiments with RDF querying showed it to be a convenient device, making conceptualising sub-graphs easier. This also required manual intervention, as the most obvious automatic generation would produce a blank node as subject, with property `siteno` and value “8864”.

5.3 Dealing with Relational Joins

In a relational database, one of the designer’s aims is to avoid redundancy by eliminating dependencies between attributes, or “normalising” the relations. In a nutshell, a normalised relation (or table) is one where each attribute depends on the primary key alone, not on any other attribute or group of attributes. (See, for example, Ramakrishnan and Gehrke (2000) or Date (2003) for full treatment of this topic.) In practice, deliberate de-normalisation is not uncommon in real databases, generally for performance reasons. Full normalisation typically forces one to create a large number of separate tables, which have to be joined at query time; it is often preferable to reduce the number of joins by accepting a certain amount of duplication. Similarly, it may sometimes be worth holding a calculated field as a fixed attribute, instead of working it out on demand each time. The sample triples listed in the last section contain a simple example. The `mapno` field, “ND27SE”, is not independent of the `ngre` and `ngrn` fields (“2902” and “7390”), and could be calculated from them given just the prefix letters, “ND”.⁵ Despite such examples, one can certainly consider normalisation to be the usual aim of good design.

Designing for RDF has slightly different considerations. Arguably, in the general case one will want to “pre-join” relations — which is akin to de-normalising — wherever possible. For RDF graphs held in the natural 3-column table for subject, property and object,⁶ every traversal from one node to another involves a self-join of the table. The object node of one statement becomes the subject node of the next in the chain, which involves an equality join between values in the object column and values in the subject column of the same table. With a large table (15.5 million rows here), one seeks to avoid making more such self-joins than is absolutely necessary. Therefore if intermediate nodes can be eliminated in a principled way, it must always be sensible

⁵The first digits, “2” and “7”, of the easting and northing give the 10km square (within the 100km “ND” square) and the second digits, “9” and “3” indicate the right hand lower quadrant, or SE, of this square. The 100km square letters are defined as part of the Ordnance Survey National Grid in the UK.

⁶Actually the more common implementation is as a 4-column table, where each statement is annotated with its parent graph ID.

to do this. Figure 5.1 provides an illustration of what this means in practice.

Figure 5.1 shows an example of the process used to convert RCHMS data into an RDF graph. Where there is an intermediary table resolving a many-to-many relationship between two database tables (*SITE* and *ARCHIVE* in this case), there is an opportunity for compression, by “pre-joining” the two parent tables and eliminating an unnecessary node between them. In a relational database the intermediary table must contain a foreign key for each of its parents, but in a graph representation this duplication can be avoided if the intermediary has no attributes of its own (i.e. if it contains only keys), by recognising that fact and implementing a special two-way link between the relevant primary keys. This saves one triple per row of the intermediary table (around 750,000 triples in this case). Even if the database schema allows for genuine attributes on the intermediary, the compression can be achieved at the instance relation level for those instances that do not themselves have the extra attributes. However, here it may be preferable to keep the redundant triples rather than introduce greater graph complexity by enlarging the set of predicates that can attach to the subject node in question. Because the direction of the link between the primary key nodes is arbitrary, the RDF query should allow for the subject and object nodes being reversed.⁷ For example, a SPARQL query over the graph in Figure 5.1, to list any archive items associated with sites, could be expressed as:

```
SELECT ?sitename ?arcitem ?arcdesc
WHERE {
  ?site name ?sitename .
  OPTIONAL {
    {{?site siteArc ?arc} UNION {?arc siteArc ?site}}
    ?arc arcType ?arcitem .
    ?arc description ?arcdesc .
  }
}
```

The `UNION` clause ensures that all `siteArc` links are found, whichever way round they point. This query will return the following results:

SITENAME	ARCITEM	ARCDISC
----------	---------	---------

⁷In practice, one would often know, because of the way the RDF graph was constructed, which way the links face.

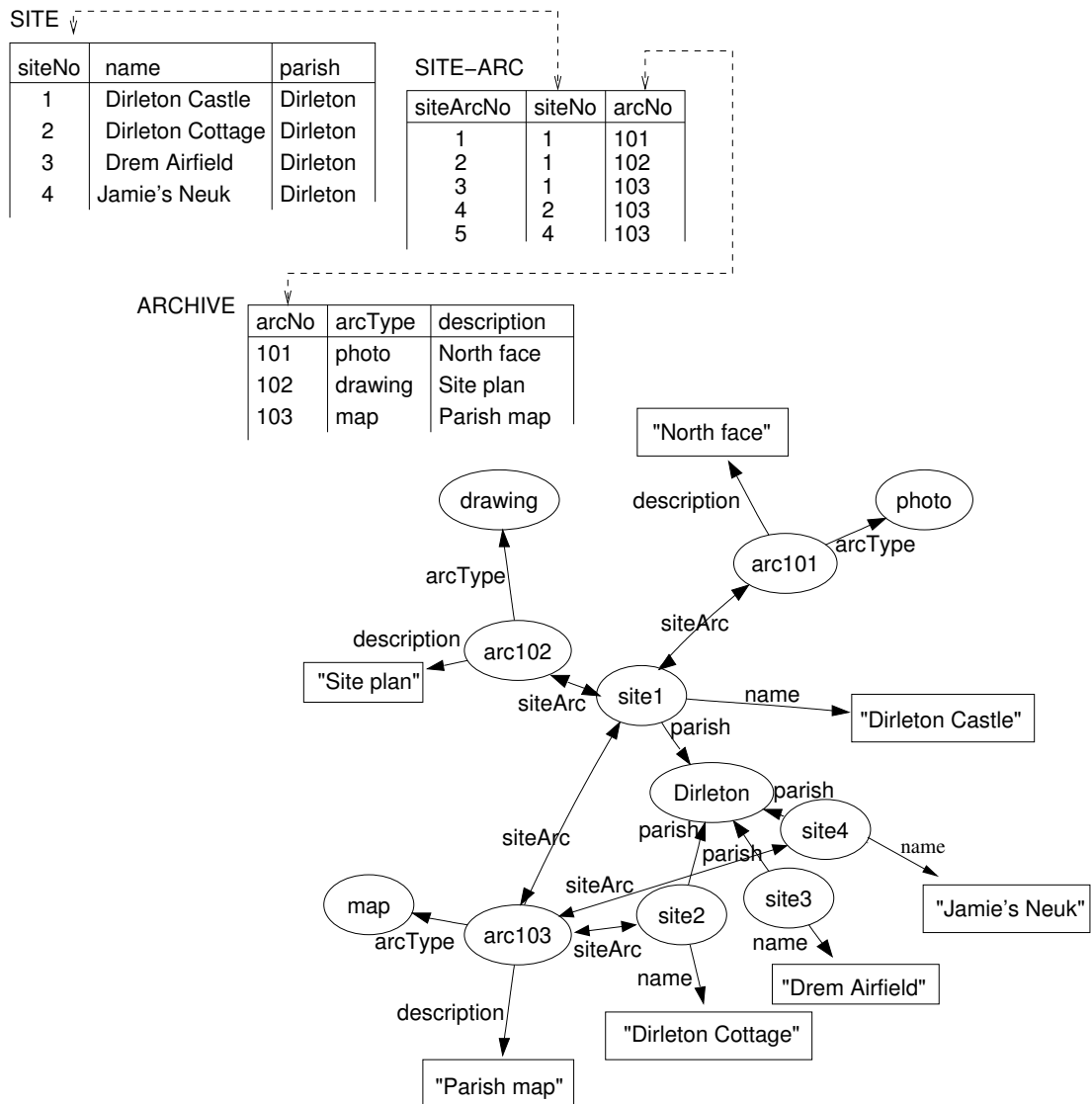


Figure 5.1: The table fragments are translated by creating one triple for each column of each row. The subject of each triple appends an ID string (*site* or *arc*) to the primary key field. The method takes advantage of the fact that the intermediary table between SITE and ARCHIVE has no local attributes.

"Dirleton Castle"	photo	"North face"
"Dirleton Castle"	drawing	"Site plan"
"Dirleton Castle"	map	"Parish map"
"Dirleton Cottage"	map	"Parish map"
"Drem Airfield"		
"Jamie's Neuk"	map	"Parish map"

In a relational database, fields with a fixed choice of values — such as `region` in the RCAHMS database, which can only take a value from a defined list — are usually implemented with codes pointing to a lookup table. When translating to RDF triples, it makes sense to resolve these values so that, once again, an unnecessary intermediate node is eliminated. This means a join to the lookup table is needed during the extraction process. If one chooses to do this, it is another instance of a manual intervention in the basic automatic conversion procedure, to add to those discussed in Section 5.2.

A final example of where an intervention, based on knowledge of the RDBMS schema, would be sensible, is in the case of concatenated keys. Figure 5.1 once again provides an illustration: the intermediary `SITE-ARC` table is based on a real table in the RCAHMS database, which actually uses a concatenation of the two foreign keys, `siteNo` and `arcNo` as its primary key. This is awkward to translate into RDF, where we want a single identifier for the resource node. Therefore a preliminary step was run in the RCAHMS database, to add an extra column to the table and generate a surrogate key (called `siteArcNo` in the diagram).

5.4 Schema Design

The informal proposal cited earlier (Berners-Lee (2006)), for RDBMS to RDF conversion, does not go into detail about what kind of schema relations would typically be implemented. On the face of it, the extensive metadata suggested does not need to be built into each resource's URI, as it might be more naturally be expressed using RDF Schema (RDFS), in properties such as `rdf:type`, `rdfs:domain` and `rdfs:range`. However, whether this saves space is another question, as the number of triples would be approximately doubled. For each distinct subject node one would need an `rdf:type` statement, and for each distinct property arc a pair of `rdfs:domain` and `rdfs:range` statements. The exact ratio depends on the number of literals (assuming these don't

also need to be explicitly typed) and on the amount of other schema information, such as `rdfs:subClassOf` and `rdfs:subPropertyOf` relations.

The schema design for *Tether* has not yet been finalised, but experiments with subsets of the data have shown the need for some manual intervention in compiling the set of properties that a node of a particular type can have. Because the final application is intended to involve producing summaries of subgraphs by property category, it is desirable that the number of different properties of, say, a `site` node is fairly small. As Table 5.1 shows, any given `site` node may have up to 50 attributes. The intention is to reduce this a much smaller number, by grouping similar attributes together and introducing some simple hierarchical relationships between them. For the RCAHMS graph a set of nine top-level properties is planned, each subdivided into up to four sub-properties (but with no lower-level divisions).

For example, here are the automatically generated statements used in earlier discussion:

```
rc:site8864 rc:ngre "2902" .
rc:site8864 rc:ngrn "7390" .
rc:site8864 rc:mapno "ND27SE" .
rc:site8864 rc:region <http://ltg.ed.ac.uk/rcdb#regionHIGHLAND> .
rc:site8864 rc:district <http://ltg.ed.ac.uk/rcdb#districtCAITHNESS>
rc:site8864 rc:parish <http://ltg.ed.ac.uk/rcdb#parishCANISBAY> .
```

In the final data extraction process, these were rewritten so that the property resources reflect the fact that all the statements are about the location of `site8864`:

```
rc:site8864 grid:ngre "2902" .
rc:site8864 grid:ngrn "7390" .
rc:site8864 grid:mapno "ND27SE" .
rc:site8864 place:region <http://ltg.ed.ac.uk/rcdb#regionHIGHLAND>
rc:site8864 place:district <http://ltg.ed.ac.uk/rcdb#districtCAITHNESS>
rc:site8864 place:parish <http://ltg.ed.ac.uk/rcdb#parishCANISBAY>
```

The `grid:` prefix denotes `<http://ltg.ed.ac.uk/rcdb/loc/grid#>` and `place:` is `<http://ltg.ed.ac.uk/rcdb/loc/place#>`.

Similar grouping was done for site classification attributes, agents such as people or organisations, date fields, indicator flags, and so forth. This step necessarily had to be done by hand, with knowledge of the RDBMS database structure.

Table Name ^a	Rows	Columns	Triples
rcSiteTable	221,416	50	3,800,380
rcSiteArcJoin	768,416	2	768,416
rcArchiveTable	680,082	46	9,789,950
rcSiteBibJoin	167,837	4	500,850
rcBibliogTable	51,302	22	554,831
Totals	1,889,053	124	15,414,427

Table 5.1: Conversion of RCAHMS data into triples. The theoretical maximum number of triples that could have been generated is the product of the number of rows and columns: 234,242,572.

^aThe table names have been changed to make their names self-explanatory.

5.5 RDF Bulk Load Performance

The database was exported using SQL scripts which wrote RDF triples in N3 format to a dump file. This step took less than 10 minutes. The output file was 680 Mb in size and contained almost 15.5 million triples. As Table 5.1 shows, the theoretical maximum number of triples, if all database fields were populated, is very much greater: more than 234 million. The ratio of possible to actual is over 15:1. This data sparsity is a feature of the domain, where recording methods have changed over the years, and the amount of detailed survey and recording has varied with the availability of funding. Thus some data items have many attributes and others very few, and the composition of the attribute sets changes significantly as recording practices evolve with time.

The data was loaded into Jena using Java utilities provided with the package. The input data file was split into manageable chunks, of around one million statements each. Inevitably there were several false starts as part of the learning process — generally when unexpected characters caused the process to crash. The Java heap size also needed to be increased; which was discovered through trial and error. Since Jena maintains the RDF graph as a *set* of statements, restarting an interrupted process is easy, as duplicate entries are automatically ignored. On the other hand, this integrity checking means the bulk load process is fairly slow: in this case the overall time to load 15.5 million triples was 5 hours and 40 minutes, a rate of roughly 45,000 rows per minute.⁸ If the RDF store is intended as the front-end for a dataset maintained in a

⁸Using a machine with a 3 GHz processor and 1 Gb RAM.

relational database, this transfer time is significant. For a constantly changing database it would probably need to be repeated daily.

Jena uses a 4-column (subject, predicate, object, graphId) table as the central core of its repository, with indexes on the object column and on the combination of subject and predicate columns. The indexes on the RCAHMS graph are 77% of the size of the data tables, and appear to be built during the loading process.⁹ On smaller graphs the ratio of index size to data is higher, the index being larger than the data tables for very small models (where of course it doesn't matter as the data volume is trivial).

5.6 Other Conversion Issues

All of the collections databases being used in this project contain a high proportion of free text. The very large text fields (with datatypes `long` or `longtext` in Oracle and MySQL) are dealt with separately, not by the methods described in this chapter. (See Chapter 7 on Relation Extraction.) But there are numerous other fields which have unstructured, descriptive entries: image captions, archive item descriptions, site name fields, and so on. Even the short fields — including some whose values are mapped to URI resources instead of literals — are frequently free-entry fields, without validation checks. Personal and organisation names are examples. In a real database such fields will inevitably contain at least a small proportion of incorrect data, generally caused by typing errors. If the value is mapped to a literal it is unlikely to cause errors during loading into the RDF model, though an exception is the ubiquitous quotation mark. Single and double quotes occur very frequently in the text fields and, since the N3 format uses double quotes as literal delimiters, these must all be escaped or converted. Where the value is destined to become a URI, the number of invalid characters is much greater, and a great deal of time was spent in dealing with characters such as `#`, `%`, `{`, `}`, `<`, and `>`, and replacing spaces with `+` signs. The ``` character (the backquote or grave accent, UTF-8 0x60) was particularly awkward to deal with properly. As explained, these characters often originate in typing errors and so their whereabouts is unpredictable. There is a trade-off to be made between examining *every* character that is exported from the database to make sure one finds them, and only running expensive `replace()` functions where they are actually necessary.

⁹This seemed to be the case during data loads observed, though the more usual procedure is to drop indexes during bulk loads and recreate them as a separate step afterwards. This is often significantly faster, and usually leads to a tidier index.

Determining whether a database field is empty and should therefore be skipped by the extraction process is not quite as straightforward as it sounds and, once again, knowledge of the data structure is needed. A text field that is blank or null can safely be ignored (though see next paragraph), but a numeric field containing “0” may or may not be providing significant data.

The source dataset for this work occupied around 380 Mb. After conversion to RDF its size was 2.7 Gb (in both cases, with indexes). This is a lower ratio than was estimated in Section 5.2, but the RCAHMS data does not match the criteria given there. The average field length is long (because of the high proportion of text “notes” fields), so the ratio of metadata to data value is lower than it would be for short fields. Secondly, the tables are unusually sparse, i.e. there are a lot of different fields but a high proportion are null for any particular item. This leads to far fewer triples being generated, because (except in a few specific cases, where the designer has allowed NULL to have a particular meaning¹⁰) a null value does not require an RDF statement to be generated. Thirdly, the URI mapping used was much less verbose than what was discussed above. Despite all these factors, the size increased by a factor of seven. In many situations, such an increase in size would be a significant disadvantage. Using full URIs feels awkward and inefficient, and there seems a strong likelihood that application designers will compress their data by stripping them and using pointers; no doubt keeping the options of exporting and importing RDF in the various standard formats (XML, N3, N-triples etc.), but not using it as a native format. If not standardised,¹¹ this would obviate a lot of the advantages that RDF promises in terms of direct access to distributed data items. Tagging information with its provenance is clearly fundamental for the Semantic Web, but it doesn’t automatically follow that URIs are the best way to do it.

5.7 Implications for Graph Querying

The conversion from RDBMS format to RDF is the first step towards creating a query application. The programme of query experiments is described in Chapter 8. The query languages used are SQL for RDBMS data and SPARQL for RDF. One of the

¹⁰See Codd and Date (1993) for a polemic on the iniquity of such design decisions. Nevertheless it happens, not infrequently.

¹¹It’s not immediately obvious why pointers that are only locally unique could not be allowed by the standard, with the parent data repository itself carrying a unique URI. The use of private IP addresses with NAT (network address translation) seems an obvious parallel.

things it would be interesting to test is whether SPARQL introduces restrictions on graph query operations.

To explain this in more detail: several authors (see Angles et al. (2004); Angles and Gutierrez (2005); Stuckenschmidt (2005)) have asserted that the standard RDF query languages, such as SPARQL, are lacking in important functionality because they do not allow graph searching operations such as finding shortest paths between nodes, comparing the degree of nodes, and performing k -neighbourhood queries. Anyanwu and Sheth (Anyanwu and Sheth (2003)) cover similar ground on the difficulties of discovering *how* RDF resources are related (i.e. the paths between them and the labels of these paths), and propose extending the current languages with their ρ -query operator¹² to provide this functionality. Navigli and Velardi (Navigli and Velardi (2003)) have shown the practical value of finding paths between pairs of graph nodes, in their *Ontolearn* system.

When an RDF store like Jena passes SPARQL queries to its database back-end, it must translate them into SQL. This raises the possibility of using SQL directly against the triples tables, in which case the SQL query is not restricted to using the smaller set of operations possible in SPARQL. The assertion is that RDF query languages disallow queries that would actually be useful. Although to test this fully one would need to use a graph database that incorporated a range of graph algorithms (GraphLog, Consens and Mendelzon (1990), is a possibility), it seems a worthwhile experiment to compare SPARQL with SQL on the same data. Operations such as finding the degree of nodes and k -neighbourhood queries where k is fixed certainly seem possible in SQL.

In addition, further work is planned on adding schema relations to the graph models. The pros and cons of alternative URI mapping procedures will also be explored in more detail.

5.8 Summary

This chapter has described a data conversion exercise on a real-world database, that turned out to involve issues not anticipated from earlier experiments with smaller sets of data. The reasons are partly to do with the inevitable presence of rogue errors in real data, assembled over years by humans. The ratio of time taken to export the RDBMS data (under 10 minutes) to that needed to import it as RDF (5 hours 40 minutes) was 1:34. Although certainly not tiny, at 380 Mb the source dataset was not a particularly

¹²The authors use the term “ ρ path” to signify an instance of a path between two nodes.

large one by today's standards. This suggests that the time needed for conversions to RDF may be such as will preclude the use of RDF as a mechanism for presenting data maintained in large RDBMS systems, unless a way is found of updating an existing graph instead of recreating it from scratch, whenever a refresh is needed to reflect changes in the source data. Such an updating mechanism might automatically compare two local sub-graphs and add or delete statements to make them match each other.

The size of the RDF dataset was also noted as a potential problem. The 380 Mb source data was translated to a 2.7 Gb RDF database, i.e. multiplying its size seven times. This is owing to the enormous amount of space consumed by repeating very similar URI strings in 15.5 million statements. The necessity for this is questioned. Furthermore, the conversion process involves decisions on whether graph nodes should be implemented as literals or resources with URIs, and these choices are not always straightforward.

In a conversion exercise like this, character set issues can be tedious and time-consuming, as characters that are prohibited in URI strings must be eliminated in some way. Standard tools to perform such conversions automatically are now becoming available.

The starting point for this exercise was that the conversion from RDBMS tables to RDF triples could be largely automated. As the chapter has shown, there proved to be a significant number of cases where manual intervention, aided by specialist knowledge of the source database schema, was required in order to make rational conversion decisions. This is perhaps not unexpected, but it suggests that world-wide conversion of relational data to RDF may not be as rapid as one might like.

Chapter 6

Named Entity Recognition

Given a goal of transforming free text material into a graph structure, the approach taken is to extract binary relations between pairs of terms in the text. Once found, these relations can readily be converted to RDF triples. To simplify the task, and because the graph nodes should represent significant content-carrying terms, the relation pair terms will all be Named Entities (NEs), which must therefore be identified and categorised, as a preliminary. This chapter describes work on this first step, of Named Entity Recognition (NER).

Text data from the RCAHMS corpus was annotated with entities and relations, as described in Chapter 4. (Section 4.3.1 covers NE annotation.) The handling of nested NEs, where one entity string contains others within it, is important and various previous approaches to nested entity discovery are considered below, before detailing the method actually used. The experimental setup and results are described and discussed later in the chapter. Finally, Section 6.5 covers the extension of the system from the relatively small annotated corpus (1,546 documents) to the whole RCAHMS dataset (250,000 documents).

6.1 Nested Entities

NER is generally treated as a classification task, where a sequence of tokens is tagged with labels by the classifier. Where entities overlap or are nested within one another, classification becomes difficult, because an individual token may require more than one label. In the RCAHMS corpus, up to three levels of nesting can occur. For example, in the string

[[[Edinburgh]^{PLACE} University]^{ORG} Library]^{ORG}

the token “Edinburgh” is a PLACE entity on its own, and part of two distinct ORG entities.

Commonly, in NER tasks, only a single level of nesting is dealt with — generally the longest string, or outermost entity. However, this NER work is in preparation for Relation Extraction, and the subject and object of each relation will be a Named Entity. If nested entities are omitted then relations using them will also be lost.

For example, in the kind of text we are dealing with, the nesting of a PLACE entity within a longer entity string is quite common (as in `[[Aberdeen] School of Architecture], [Earl of [Argyll]]`), and so forth). Although the resulting *hasLocation* relations will probably be important in query applications, they are likely to be missed unless we can deal with nested NEs. Similarly, bibliographic references (which are classed as EVENTS with subclass DESCRIPTION) typically contain PERSNAME and DATE entities which may participate in separate relations. A user who is interested in historical sites mentioned in a given bibliographic work (such as a paper by a particular archaeologist), is very likely also to be interested in sites associated with the author of that work, which will probably date from the same period, or be similar in some other way. Thus the discovery of *all* entities, regardless of level, is important. Various methods documented in the literature will be examined before going on to describe my experiments using a novel approach to the problem.

Interest in nested NE detection has increased in recent years, though it is still the case that most NER work deals with only one level at a time. One way of dealing with nested entities (Zhou et al., 2004) is to detect one level (the innermost in this case) and then derive rules with which to find other NEs containing these as substrings. The dataset used was the GENIA corpus (Collier et al., 1999). The authors report an improvement of around 3% in the F-score under certain conditions.

A different approach (McDonald et al., 2005) uses structured multilabel classification to deal with overlapping and discontinuous entities. (The corpus of MEDLINE abstracts used did not contain nested entities.) In multilabel classification, each example is associated with a set of labels instead of just one. Here, the labels are structured in the sense that they do not come from a pre-defined set but are built for the instance in hand. Theoretically, the number of different labels is exponential on the length of the instance, but the set for consideration can be limited using the structure of the labels. The method was successful when compared with standard sequential tagging, such as is used by the CandC classifier (Curran and Clark, 2003) employed in this work.

In another study (Gu, 2006), the problem is cast as a binary classification task, using a one-vs-rest scheme, to get round the difficulty of individual tokens requiring more than one label if they are part of a nested entity. In this work just two entity classes (proteins and DNA in the GENIA corpus) were used, in separate experimental runs, with two levels of nesting: outermost and any one inner. The study found that their “outmost labeling” method recognised outermost entities better, and “inner labeling” was better for inner NEs (as one might expect).

The idea of using “joined label tagging” (Alex et al., 2007), in which the number of entity classes is expanded to include concatenations of overlapping class labels, is discussed in Section 6.2 below. It is contrasted with the method proposed here, which concatenates tokens instead, so that each nested entity string has its own separate label. Alex et al. also use cascading and layering methods in a pipeline combining taggers, to achieve good results for nested entity discovery.

6.2 Experimental Setup

The 1,546 text documents comprising the corpus were tokenised, split into sentences and POS tagged, before being formatted for annotation using the MMAX2 annotation tool.¹ The data was then reformatted for the CandC maximum entropy classifier (Curran and Clark, 2003), using the BIO notation. The beginning of an entity string is given a “B-” prefix before its label, tokens within the entity string have an “I-” prefix, and tokens that are not entities are labelled “O”. Thus, a phrase like “...in the National Monuments Record” becomes “in_O the_O National_B-ORG Monuments-I-ORG Record_I-ORG”. Tagging was done using 10-fold cross-validation.

The distribution of NE types is summarised in Table 6.1. In total, the annotated corpus contains 28,272 entity strings, if all levels of nesting are included, and all lengths of entity string. For reasons explained below, NE strings consisting of seven or more tokens were excluded, bringing the total down to 27,453. The proportion of NEs having other entities nested within them is 9.4%, whilst 18.7% are nested within longer NEs. Each containing entity typically has either one, two or three shorter NEs within, with two being the commonest. The corpus contains no entities with more than three levels of nesting, i.e. at most we have outer, inner and innermost levels. No disjoint entities (where the tokens comprising the NE string are non-adjacent) were included.

¹<http://www.eml-research.de/english/research/nlp/download/mmax.php>

Entity Type	Raw Count	≥ 7 tokens	Kept
SITETYPE	5,675	7	5,668
ADDRESS	3,558	100	3,458
EVENT	3,843	667	3,176
DATE	3,520	1	3,519
ORG	2,737	7	2,730
SITENAME	2,737	25	2,712
PLACE	2,509	6	2,503
PERSNAME	2,318	0	2,318
ARTEFACT	879	0	879
PERIOD	406	6	400
ROLE	90	0	90
Total:	28,272	819	27,453

Table 6.1: NE types distribution

As discussed above, the problem with nested entities is that individual tokens require multiple labels if they participate in more than one entity string. One possibility is to concatenate labels; for the “Edinburgh University Library” example cited above, the first token might be labelled B-PLACE_B-ORG_B-ORG, the second O_I-ORG_I-ORG and the third O_O_I-ORG. This makes the task more difficult because the number of categories to choose from is larger, while the number of training instances remains the same, but it enables all levels of entity to be recognised. This joined label tagging technique has been successfully used in the biomedical domain (Alex et al., 2007) and that work may be extended to cover the RCAHMS dataset, which would enable a comparison between joined label tagging and the technique used here.

The alternative tried here is to concatenate the tokens instead, so that each entity string becomes a single token and can be given its own correct label. To achieve this, a maximum entity string length must be determined in advance, and then every token in the corpus is concatenated with those following, up to the chosen length. For example, if the maximum entity length to search for were 3, then the phrase “when Edinburgh University Library was built” would be tokenised and labelled as follows:

when O
 when_Edinburgh O
 when_Edinburgh_University O
 Edinburgh PLACE
 Edinburgh_University ORG
 Edinburgh_University_Library ORG
 University O
 University_Library O
 University_Library_was O
 Library O
 Library_was O
 Library_was_built O
 ... and so on.

An analysis of the distribution of entity string lengths showed that 97.10% were of length 6 tokens or fewer, though the longest was 25 tokens. The maximum length figure was therefore set to 6 for this experiment, which excluded 819 NEs. This is consistent with the overall goal of the project, because long strings are very unlikely as user query terms. On the face of it, the advantage of this method is that the number of categories is unchanged, while the amount of training data is increased — though the big increase is in the number of negative examples. The obvious drawback is the increased time taken for training the classifier, and also that (depending on how the tokens are arranged) the sentence length is increased 6-fold.

The training data was presented to the classifier in the format shown above, with features added. For each individual token a set of 6 “multi-tokens” was generated, with one token in the first, two in the second, and so on. The test data was in exactly the same form, without the tag labels. The classifier output a single tag per token received: from the set of 11 class labels plus “O”.

The final token of the concatenated string was made salient to the classifier by passing it the POS tag of the last token in the string. The final token was picked each time for consistency and because it is most likely to be the head word of any entity string. The following unigram features were also experimented with, but produced only a marginal improvement in overall performance (see Table 6.2, run 10):

- position within set of 6 multi-word tokens: p1 to p6
- contains “_”: yUnd or nUnd

- capital following “_”: ic0 (none), ic1 (some) or ic2 (all)
- last token type: letters converted to “A” or “a”, digits to “0”, punctuation unchanged (so “Shetland” becomes “Aaaaaaaa”).

The CandC classifier is, naturally enough, optimised for standard sentences. It has a number of built-in features based on the bigram and trigram context in which each token appears, and it also makes use of gazetteers of personal names and place-names. The multi-word tokenisation may well confuse the classifier and, to explore the multi-word method thoroughly, a classifier would have to be configured with it in mind. Therefore some experiments were run with “previous word” and “previous POS tag” bigram and trigram features, where “previous” refers to the preceding single token from the original corpus text, and not the immediately preceding multi-word token. Similarly, forward bigrams and trigrams were also tried. These features were chosen as they are known to be particularly useful to a standard NE classifier, but there is scope for further exploration.

The experiments were run almost exclusively with the CandC experimental NE tagger, which is a state of the art system, tuned for NER over English text. Since it was realised that the multi-token method would be penalised by not in fact being very like normal English text, a simple experiment was done with another, general purpose tagger, Zhang Le’s Maximum Entropy Toolkit, or ZLMaxent (Zhang and Yao, 2003). The aim was not to tune this tagger properly for NER, but merely to compare the single and multi-token methods on a level playing field, where neither had any advantage.

6.3 Results

Results so far indicate that the multi-word tokenisation technique can improve the tagger’s performance, when combined with the extra word and POS features mentioned above.

Table 6.2 summarises the overall NER results, comparing a number of runs of the standard single-token method with multi-word tokenisation, and showing Precision, Recall and F-score (harmonic mean) figures, calculated using the CONLL scorer. For the single token runs, only one level of entities is available each time. The table shows what the total number of available entities was for each experiment, and the scores are percentages against this total. The scores for the multi-token runs are percentages of

Run	Description	Precision %	Recall %	F-score %	CorrectNEs
Single-token: average 24,448 NEs (varies because of random selection)					
1	Single tok, all lengths, rand1	70.47	69.73	70.10	16,923
2	Single tok, all lengths, rand2	71.05	69.75	70.39	16,918
3	Single tok, outermost NEs	74.77	72.42	73.58	16,671
4	Single tok, maxlen 6, random-1	74.64	72.67	73.64	17,931
5	Single tok, maxlen 6, random-2	74.57	72.65	73.60	17,920
6	Single tok, outermost, maxlen 6	77.06	75.09	76.06	18,359
7	As run 6 + domain gazetteers	76.98	75.18	76.07	18,379
Multi-token: 27,453 NEs available					
8	Multi-tok, basic	80.75	65.24	72.17	17,899
9	Multi-tok, domain gazetteers	80.52	65.67	72.34	18,015
10	Multi-tok, unigram features	82.14	66.79	73.67	18,322
11	Multi-tok, POS+word trigrams	81.84	66.26	73.23	18,178
12	Multi-tok, w_{i-1} alone	87.70	66.79	75.83	18,322
13	As 12 with weights adjusted	84.79	70.81	77.17	19,426
14	As 12 with weights adjusted	82.96	73.39	77.32	19,860
15	As 12 with weights adjusted	78.43	75.91	77.15	20,825

Table 6.2: Summary of NER results

the full set of 27,453 NEs, as this method is capable of training on, and outputting, all the NEs at once.

The first run included all NEs, selecting a single one at random wherever there was a nested set. Run 2 is exactly the same, with a different random selection. It was unclear whether this would be a fairer baseline than using only the outermost entities, as is commonly done, so run 3 was included. The significant difference in performance was unexpected: consistently using outer entities produced a 3% improvement in the overall F-score, whereas one might expect such entities to be much harder to recognise, as they are sparser in the corpus. Runs 4 and 5 (which are the same except for using a different random selection in each family of nested entities) excluded the longer entities — those consisting of strings of more than 6 tokens — in order to match the multi-word method, which also excludes them. The results are not significantly different from the previous run, which supports a conclusion that the classifier is sensitive to

the length of the entity strings, but that it is a mixture of lengths (as in runs 1 and 2) that causes problems, rather than their absolute length. To test this, run 6 used only outermost NEs (as in run 3) and excluded the long ones. This produced another significant performance gain, to a F-score of 76.06%. This suggests the possibility of improving the tagger's model by training it separately on entities of each different string length.

The final run of the single-token set used extra gazetteers, in addition to those built-in to the classifier (for personal names and place names). The main ingredient was the Thesaurus of Monument Types (TMT), which is based on MIDAS (MIDAS, 2003). Terms from the Thesaurus of Object Types, maintained by MDA (which formerly stood for "Museum Documentation Association") and available from the same source as TMT, were also added. These were intended to help the SITETYPE and ARTEFACT classes respectively. This produced no improvement, which is not altogether surprising, as work in the NER field has shown that gazetteers tend not to enhance performance significantly if the training data is sufficiently representative (Mikheev et al., 1999). The main reason for including them here was in order to test whether having multi-word terms helped. With a multi-word tokenisation it was clear that longer strings could be included in the gazetteer list, by concatenating them (with underscore characters) exactly as the corpus tokens were concatenated. This is a simpler approach than those needed to handle multi-word gazetteer entries in a standard single token setup.

Runs 8 to 15 used the multi-word tokenisation. The first, the simplest for this approach, used no additional features and no gazetteers apart from the classifier's own. It performed worse overall than the single-word method (though on a larger number of NEs), but it seems likely there was a mixture of positive and negative effects (see below). Adding the domain gazetteers (run 9) had no impact, answering the question raised about multi-word gazetteer terms. This is not especially unexpected: gazetteers tend only to include terms the classifier sees plenty of examples of anyway, and has little trouble with. It may be possible to improve the contribution made by gazetteers, by training a separate model for them, as recent work has shown (Smith and Osborne, 2006).

The unigram features discussed in Section 6.2 were tested in run 10, and produced a small improvement. It might be worth further experimentation to find better features characterising the multi-word token in a useful way.

A series of runs was made, of which run 11 and run 12 are shown as being the most noteworthy, using various combinations of previous and next POS tags and words:

$pos_{i-2}, w_{i-2}, pos_{i-1}, w_{i-1}, pos_{i+1}, w_{i+1}, pos_{i+2}, w_{i+2}$. This was an attempt to reproduce some of the normal features built-in to an NE classifier. Surprisingly, w_{i-1} (previous word) was not only by far the most useful (run 12), but was much better on its own than when combined with the others (run 11 used a combination of all the POS and word features just listed).

It is noticeable that precision is improved at the expense of recall in the multi-word experiments. For this domain, that is arguably a benefit, as is discussed below. However, in order to try to balance precision and recall better, a series of trials was carried out with varying weights for each class applied to the maximum entropy model that CandC uses. This adds constraints that the model must fit, and biases it towards or against selecting a particular NE class tag. Runs 13, 14 and 15 are examples of slightly different weightings that each improved recall without losing too much precision, and improved the overall F-score. In run 13 the weighting favours SITETYPE and EVENT classes, which are large and important classes having poor recall (see Table 6.3). In Run 14 the weighting is uniform across all the NE classes but biased against the “O” class, to improve recall across the whole set by having a less cautious model. Run 15 is similar to 14 but with an even more smoothing, and almost balances precision and recall. The same trials were made for the single word models where, as expected (since these are already well balanced), no significant improvement in overall score was possible.

In some respects the multi-word tokenisation must surely have a negative impact. As already mentioned, the tagger has built-in bigram and trigram features that will be skewed in these experiments. Also, the tagger makes use of prior knowledge of word frequencies derived from large English text corpora (1 billion words), and at least 5/6^{ths} of the tokens will appear as unknown words. Therefore the fact that a slight improvement in overall performance is possible suggests that there must be a definite positive effect balancing the negative.

As is usual, the results for precision, recall and F-score are given as percentages. This is slightly misleading, as the total number of entity strings is higher in the multi-word experiments than in the single baseline where only one of each nested entity set is available. For runs 1 to 7 the average number of NEs that could be found is 24,448. (The total varies for each of the random runs, from 23,020 to 24,675, because a long entity string may contain several shorter ones.) For runs 8 to 14 it is the total NE population: 27,453. Thus the multi-token method can output an average of over 12% more NEs than the single-token method. The actual number of correctly predicted NEs

	Run 7 (best single)			Run 12 (multi)			Run 15 (12 smoothed)		
NE class	P %	R %	F %	P %	R %	F %	P %	R %	F %
ADDRESS	79.99	82.03	81.00	82.32	83.42	82.87	70.97	85.50	77.56
ARTEFACT	53.41	32.49	40.40	71.14	16.29	26.51	56.62	29.73	38.98
DATE	86.38	92.04	89.12	95.09	82.01	88.06	88.22	90.68	89.43
EVENT	85.24	73.43	78.89	94.81	64.47	76.75	87.74	76.26	81.60
ORG	91.23	90.98	91.11	99.27	89.22	93.97	98.30	91.24	94.64
PERIOD	64.59	56.61	60.34	83.26	44.75	58.21	72.29	63.25	67.47
PERSNAME	83.49	84.69	84.08	96.86	77.13	85.87	91.26	85.15	88.10
PLACE	83.34	78.15	80.66	94.88	66.69	78.33	91.17	69.77	79.05
ROLE	93.10	60.67	73.47	98.00	54.44	70.00	96.15	55.56	70.42
SITENAME	62.53	71.04	66.51	65.98	62.60	64.24	53.80	69.46	60.63
SITETYPE	67.32	64.88	66.08	82.17	45.07	58.21	71.52	63.70	67.38
	76.98	75.18	76.07	87.70	66.79	75.83	78.43	75.91	77.15

Table 6.3: Detailed NER results

for each run is included in the last column of Table 6.2. If one were to compare the results on the total NE population (i.e. deeming the single-token model to have missed all the NEs unavailable to it), then the best single-token run (number 7) would have recall of only 66.95%.

Table 6.3 gives the detailed results for precision, recall and F-score within each NE class, for runs 7, 12 and 15. Numbers 7 and 12 are the best from each method with standard constraints on the maximum entropy model, and 15 is the same as 12 but with smoothing applied to balance precision and recall as closely as possible.

As noted, precision is much improved by the multi-word technique, whilst recall suffers. The ARTEFACT class performs poorly in all variants, but particularly with the multi-word tokens. It is one of the sparser classes and the members do not seem to be sufficiently distinctive to be easy to model. The scores for ROLE are probably not very reliable, as this class is tiny (see Table 6.1). With smoothing, the multi-word model produces a slightly better F-score than the single-token method (77.15–77.32 compared to 76.07), whilst outputting considerably more NEs: 20,825 as against 18,379 — or 13.3% more.

Analysis of the errors made by the multi-token classifier is interesting. In calculat-

GOLD STANDARD	CLASSIFIER ERROR	COMMENT
J I McKinley PERSNAME	I McKinley PERSNAME	<i>partial string tagged</i>
Dental School SITENAME	Dental School ADDRESS	<i>wrong class, but ok for querying</i>
National Library of Scotland SITENAME	National Library of Scotland ORG	<i>as above</i>
St Abb's Head PLACE	St Abb's Head SITENAME	<i>as above</i>
London Mint O	London Mint ADDRESS	<i>classifier choice seems better</i>
three of which SITETYPE	<i>tagged as O</i>	<i>co-reference not handled well</i>
dyke SITETYPE	this dyke SITETYPE	<i>determiner wrongly included</i>
the enclosure SITETYPE	enclosure SITETYPE	<i>determiner wrongly omitted</i>
East Cults Parish Church SITENAME	Cults Parish Church SITENAME Parish Church SITETYPE	<i>2 errors, but ok for querying</i>

Table 6.4: Examples of NER errors

ing the error percentages, all deviations from the gold standard are counted as errors; but in practice some are more significant than others. Table 6.4 lists some examples of common classification errors that, from the point of view of building a graph of relationships between entities, would not be very harmful. Unfortunately, it is difficult to calculate what percentage of the errors are like these, as opposed to more damaging failures. Including or excluding extraneous tokens (such as a preceding determiner) is a pity, but much less serious than missing an entity altogether. Similarly, it is more important to detect the presence of an entity, or “content carrying term” than to classify it correctly: Table 6.5 compares “unlabelled” results corresponding to Table 6.3’s, where the scoring is just on presence or absence of an NE as detected by those models (without retraining). It might be interesting, given more time, to retrain the models for different NE class sets, perhaps merging the locational classes (PLACE, ADDRESS, SITENAME) together, and the time-based ones (PERIOD, DATE), that were noted in Chapter 4 as likely to be hard to distinguish.

Run	Precision	Recall	F-score	Correct NEs
7	83.54	81.82	82.67	18,906
12	93.07	71.32	80.76	18,986
15	85.14	82.46	83.78	21,952

Table 6.5: “Unlabelled” results for Table 6.3 runs

	P %	R %	F %
single-word tokens	41.06	48.56	44.49
multi-word tokens	78.59	46.90	58.75

Table 6.6: Comparison using ZLMaxent

Finally, as was mentioned at the end of Section 6.2, a simple experiment was performed using the ZLMaxent tagger, which is a general purpose maximum entropy classifier, with no NE tuning. It treats the data simply as a collection of instances with attached features, rather than as sentences made up of sequences of tokens. For this kind of classifier, the multi-word tokens are at no disadvantage, since there is no background knowledge of English word frequencies and so forth. A comparison was made using the baseline set of multi-word tokens (all 27,453 NEs that are 6 tokens long or shorter) against the corresponding single-word set (as used for run 6 in Table 6.2, having 24,500 NEs), with the same minimal set of features in each case. The results are shown in Table 6.6. The scores are very low, as is to be expected for a completely untuned system, but those for the multi-word tokens are a good deal higher, and the bias towards precision is strongly confirmed.

6.4 Discussion

This method for nested entity recognition appears to be promising, though further exploration would be needed to test the limits of performance that could be achieved. The drawback is the longer time needed to train the classifier, but the significant advantage of the multi-token system is that the classifier can output *all* the NEs in the corpus, instead of only one of each family of nested entities. The depth of nesting is immaterial. In the experiments described, 13.3% more NEs were correctly found by

the multi-word model as compared with the baseline single token method.

The maximum length of entity string to search for must be determined in advance — in this case a maximum length of 6 was chosen, which excludes only 2.9% of entities in this corpus. Arguably, excluding long entities is a sensible tactic anyway for this project, which aims to build a queryable graph from text relations, so entity strings can be considered as candidate query terms. There is likely to be a performance gain from dropping the longer entity strings, and they are improbable as end-user query terms.

The method increases precision at the expense of recall. For a system ultimately intended to deal with queries over cultural data by non-specialists being able to achieve high precision is good news. This type of user is generally not greatly concerned with recall — he or she does not need to see *every* example of a long barrow in Scotland — but precision is crucial. It would be a bad mistake to tell such a user that a long barrow exists where it does not. Naturally, specialists in the fields of archaeology or architectural history (the topics this data covers) may be interested in good recall as well, but this programme of work is specifically directed towards assisting non-experts, for whom trustworthy information is much more important than complete coverage.

6.5 NER Across the Entire Dataset

This section will discuss issues around extending the entity recognition across a collection of about 250,000 documents. Clearly accuracy cannot be measured over the whole RCAHMS database, but randomly chosen samples will be examined to verify that the system appears to perform as expected. The processing time needed to deal with a large collection is of interest, as being able to conduct NER on very large scales is an underlying premise of the whole project.

Chapter 7

Relation Extraction

This is the main chunk of outstanding work still to do. The chapter will describe the RE setup and results, and the conversion of extracted relations to RDF triples that can be merged with the rest of the graph data.

Chapter 8

Graph Queries

This chapter will describe the experimental work on the generated datagraph. There will be two themes in the experiments: firstly to assess RDF as a viable competitor to standard RDBMS systems in terms of basic query performance, and secondly to examine whether extending the database with relations derived from text actually enhances query power.

Comparable performance, in terms of power and speed, has to be available from RDF queries, or users are very unlikely to switch from the RDBMS query systems they have been using for many years. Section 8.1 deals with tests of this aspect.

Section 8.2 covers the second theme, describing experiments that compare queries over the basic graph derived from RDBMS fixed fields, with queries over the same graph extended with relations derived from associated text documents.

8.1 RDF Compared with RDBMS

This section will describe a series of experiments comparing querying over an RDF graph of around 15 million triples with querying over the equivalent RDBMS tables from which the graph was derived. Two particular aspects are examined: the scope or power of RDF graph queries compared with relational database queries using SQL, and the relative performance in terms of response time.

The source material is currently held as a mixture of fixed database fields, free text documents and domain thesauri (see Chapter 4). All of this material will be translated into a graph database, in the case of the free text by extracting binary relations from the documents using Natural Language Processing (NLP) techniques (see Chapters 6 and 7). Once the graph structure is in place, an interactive query application (named

Tether) will be implemented over it. See Byrne (2006), included at Appendix A, for details.

The translation of a moderately large relational database into a graph structure that can be expressed in RDF is a non-trivial exercise. Although a fully automatic transformation process would have been preferred, in practice it proved necessary to introduce some manual schema design, based on knowledge of the data content, as is discussed in Chapter 5. The consequences for query construction will be explained.

8.1.1 Experiments

This section will describe a series of query experiments carried out once the relational data was translated into a graph of 15.4 million RDF triples, held in Jena¹ with a MySQL database for persistent storage. Jena was chosen after a survey of the various RDF stores available; see HP (2004) for a description of Jena's design. The aim of the experiments is to assess the viability of RDF as a format for managing large real-world datasets, where the primary purpose of the data collection is educational, and easy access by non-specialist users is a fundamental goal.

The query language chosen for the experiments is SPARQL,² and the objectives are as follows:

1. To compare the performance of SPARQL over RDF with SQL over a relational database, for standard queries that are considered routine in the source database.
2. To examine queries that are difficult in a relational database, but may be easier in RDF. For example, the detection of implicit relationships between nodes that are within a certain distance of each other (say, with a path distance of not more than three edges between them) but where the precise nature of the relationship (the predicate or property type) may be unknown.
3. To explore problems arising from the limitations of SPARQL as regards standard graph searching operations like finding shortest paths between nodes and degree of nodes. Such operations may be required in *Tether*, as one of the aims is to provide context to the user by summarising results. This will involve counting related nodes within defined categories.

¹<http://jena.sourceforge.net/>

²<http://www.w3.org/TR/rdf-sparql-query/>

The limitations just mentioned are described in Angles and Gutierrez (2005) and Stuckenschmidt (2005). One of the experimental goals is to assess whether these theoretical restrictions can be shown to present real practical problems or not. The authors of Navigli and Velardi (2003) have shown the value of finding paths between pairs of graph nodes, in their *Ontolearn* system, though the system configuration was different from what *Tether's* will be.

8.1.2 Results

This section will present the results. In the case of the tests of performance, results will be stated in terms of the time taken to execute queries and to return results. For the tests of query power, quantitative measurement is more difficult. The standard precision and recall measures do not apply. For a large database, measuring recall is problematic — in general it is not possible to say whether all possible correct answers have been returned. As for precision: for languages like SQL and SPARQL, which partition the data into query matches and non-matches, precision is always 100%, almost by definition. Therefore the experiments are designed to produce either success or failure: either the query is possible and returns valid results, or it is impossible. Where this distinction cannot be achieved, the measure may be in terms of performance time: for example showing that a query is possible with SQL, but the performance is necessarily very poor compared with an equivalent in SPARQL.

8.2 Extending the Graph with Relations from Text

This section will describe the second series of experiments on graph querying. The RDF graph deriving from RDBMS fixed fields, used in Section 8.1 above, will now be used for comparison trials against the same graph extended by adding in the relations derived from text. The two graphs will be referred to as “basic” and “extended”.

8.2.1 Experiments

The objectives are:

1. To test whether the extended graph can be shown to produce extra information for the user. One would need to demonstrate queries that fail or produce unhelpful results against the basic graph but extract useful and valid information

from the extended graph. Of course, this does not imply that *every* query would get better results from the extended graph, but improving even a subset of query results could be considered a success. Proving the negative, i.e. that no queries are degraded by using the extended graph, is likely to be impossible.³

2. To produce summarised results that break down large result sets by category, to make them easier for an inexperienced user to understand. This procedure is described in detail in Appendix A.
3. To demonstrate the possibility of guiding queries using query expansion techniques. This is also covered at Appendix A.
4. If time permits (see caveat in Chapter 9), to test cross-domain querying. This would involve testing queries designed to extract related information from, say, both the RCAHMS and NMS datasets.

8.2.2 Results

The results criteria will be similar to those discussed in Section 8.1.2, except that there may inevitably be more subjectivity involved. In some cases it will be necessary to have human judges — ideally domain experts — to decide whether a result is valid or not. In the case of results derived from database fields, every well-formed result can be assumed to be correct; but where the information derives from extracted relations it may contain false statements because of errors in the extraction process. In most cases it will be possible to check results by comparing them against the relevant source documents.

Clearly the aim will be to derive results that are as measurable and objective as possible. However, alongside this formal academic goal is the question of whether the work may genuinely be considered useful in a heritage data management context. For this, the real criterion is whether the graph database produced is easy to create, maintain, query and generally work with, and whether it seems capable (perhaps after further enhancement) of producing better results than previously possible. These considerations are not susceptible of precise measurement — but my suspicion is that the answer will actually be fairly obvious to experienced data managers.

³It would probably involve consistency checking across the graph, and this is not expected to be feasible, partly because the data very probably contains logical inconsistencies, as “real-world” text often does.

Chapter 9

Extending to Related Domains

This chapter will cover extension of the work done with RCAHMS data to the NMS and NLS datasets. Two small datasets were annotated to the same scheme as the RCAHMS one, for testing the NER and RE steps. (Not enough data was annotated for retraining in each domain.)

If time permits, I would like to create full graph databases for each of these domains — involving a RDBMS-to-RDF step as well as NER and RE across the whole of each dataset (i.e. not just the annotated test sets) — in order to test cross-domain graph querying.

If time is short, I will content myself with producing results for NER and RE on the new domains, which will give an indication of how likely it is that the full exercise would be successful.

Chapter 10

Conclusions

This chapter will summarise and pull together the significant results, and discuss them from several aspects:

- academic contribution
- application to heritage data management
- relevance in a Semantic Web context.

It will also discuss possible extensions to the work, such as:

- natural language generation from RDF
- relation extraction as a component of text summarisation systems
- integration of spatial and graphical data
- practical application development using Web Services and agent software.

Appendix A

Tethering Cultural Data with RDF

This paper describes a research project leading to the development of a data querying application based on the Jena RDF store. The starting point is a collection of relational databases holding cultural heritage material from the National Collections of Scotland. This source data is a mixture of fixed fields and free text, supported by background material such as domain thesauri. The aim is to translate all the relevant material into RDF triples, as a step towards building a query application aimed at non-expert users who do not know how the data is structured and are ignorant of the specialist domain terminology needed to write good queries. The paper describes how the translation to RDF addresses both of these problems. Two core tasks are involved: extraction of two-place relations from free text using Natural Language Processing (NLP) techniques, and automatic assembly of all the relevant data into a graph database. An interactive query interface is proposed, in which a tailored summary based on the user's initial query is generated, and the user is then invited to refine the query based on this information.

Introduction

The programme of work this paper describes was inspired by the need to improve public access to the collections held by cultural heritage organisations in Scotland. These collections describe historic sites, buildings and objects, and comprise a mixture of structured data, text, and graphical material such as photographs, maps, plans and so forth. Although digitising the entire collections will take many years, we are now at the stage where all the text-based data is on computers along with a growing body of accompanying graphical archive. Naturally there is much interest in improving access to the material over the Internet, but although there are many query interfaces available, they are not ideal: to get the best out of them the user needs prior understanding of the

domain terminology and a grasp of how the collections are structured. Although much of the useful information is in text fields, query access to free text ranges from awkward to impossible. Another problem is that the material has been collected over decades or even centuries, and in the past was aimed at specialists. Transforming it into something accessible to the general public is a difficult and potentially very expensive task. The aim of the project described here is to explore how Natural Language Processing (NLP) techniques and ideas from the Semantic Web can bring us closer to the goal of making the information easily available to interested members of the general public, who are now the core audience. Three of the National Collection bodies have contributed copies of their data for the project: RCAHMS,¹ NMS,² and NLS.³ It was important to have a selection of datasets because an additional project goal is to facilitate cross-collection querying so that, for example, a user who is interested in a particular archaeological site from the RCAHMS database could be offered information on objects found at it, from the NMS collections.

One of the key problems is to find a way of representing the information expressed in the free text notes so that it can be easily queried alongside fixed-field data. This will be dealt with as a relation extraction task, where a relation is defined as a two-place predicate or (subject, predicate, object) triple, such as (V G Childe, excavated, Skara Brae).⁴ The fixed-field and thesaurus data is similarly transformed into triples, which are then held in a triple store repository. The data transformation tasks are described in Section A.

Once all the relevant information has been translated into the simple, standardised format of an RDF graph, it becomes possible to create a query mechanism that can guide the user through the data by exploring potentially useful links and summarising across subgraphs. Data presentation is another key aspect of accessibility — it may be a non-trivial exercise to produce clear and readable information for the user. The use of RDF suggests the possibility of using natural language generation to convert RDF statements into sentences. Section A deals with these application issues. Finally, Section A gives a brief review and conclusion.

At the time of writing, the project is in the early stages of transforming the fixed fields and thesauri. This document outlines the detailed planning that has been done

¹The Royal Commission on the Ancient and Historical Monuments of Scotland, <http://www.rcahms.gov.uk/>.

²National Museums of Scotland, <http://www.nms.ac.uk/>.

³The National Library of Scotland, <http://www.nls.uk/>.

⁴The physical implementation will be in RDF, so the subject and predicate will in fact be URIs.

for the whole programme of work. The system has been named *Tether*.⁵

Translation to RDF

As explained above, all relevant source information is to be translated into a single format of binary predicates. The obvious physical implementation of this is as RDF triples, and the resulting new database will be a collection of RDF graphs. There are interesting issues surrounding data maintenance and updating but they will not be covered in this paper, where the assumption is that the data is read-only. There are many triple stores now available; Jena⁶ was chosen as the implementation platform after a survey of a number of alternatives.⁷

The project data consists of approximately 250,000 site records and texts from RCAHMS with 750,000 associated archive item records, plus 114,000 archaeology database records with associated texts from NMS, and 20,000 records and texts from NLS covering historical collections including books, broadsheets, photographs and maps. Experiments so far suggest this will translate to something in the order of 20 million triples. It would certainly be possible to reduce the volume, but part of the project's aim is to assess the viability of the whole approach with real-world data and realistic database sizes. In choosing a triple store the link to persistent database storage (in this case in MySQL) was an important criterion. It might be possible to hold the entire *Tether* database in memory, but typically this will not be the case for real data collections, and arguably the more interesting Semantic Web applications are those capable of identifying relevant subsets of distributed stored information efficiently and loading them into memory for detailed processing.

The word “ontology” is sometimes used to refer to the type of structure being described, but here the term “graph database” will be used, reserving “ontology” for more formal systems with rulesets, over which it is possible to run logic processors and reasoners. At this stage of the project it is not clear how feasible or indeed useful it will be to attempt, for example, consistency checking across the structure.

⁵“Tether” is a dialect word for “three”, used in the north of England for counting sheep.

⁶<http://jena.sourceforge.net/>

⁷The key considerations were the use of persistent storage, the design of the triples tables and the likely development and maintenance of the system.

There are three elements to the construction of the triples database, dealing respectively with database fields, thesauri and free text.

RDBMS to RDF

At its simplest, translating a relational database to RDF triples is a straightforward process, easily automated. Every table or relation in the database can be translated into a set of triples in the form `(relationID, attribute, value)`, where `relationID` indicates the parent table, `attribute` is an attribute or column name, and `value` is the value held in that column. Thus a ten-column database table would generate ten triples per row of data. See Berners-Lee (2006) for a discussion of the issues, particularly of how to generate the URIs.

For this project some minor variations to the basic procedure are proposed, firstly to “pre-join” or denormalise the tables where appropriate, and secondly to introduce some manual schema design by grouping similar attributes (predicates in RDF) together. In addition, empty fields are omitted, and concatenated database keys are replaced by new surrogate keys.⁸

Figure A.1 illustrates the translation procedure for some simplified extracts from RCAHMS database tables. (In this diagram and the others, simple labels are shown instead of URIs, for readability.) Note that where there is an intermediary table resolving a many-to-many relationship between two database tables (SITE and ARCHIVE in this case), there is an opportunity for compression, which Figure A.1 illustrates. In a relational database the intermediary must contain a foreign key for each of its parent tables, but in a graph representation this duplication can be avoided if the intermediary has no attributes of its own (i.e. if it contains only keys), by recognising that fact and implementing a special two-way link⁹ between the relevant primary keys. This saves one triple per row of the intermediary table (around 750,000 triples in *Tether*). In practice, because of the way the data was loaded, one would probably know which way round the links point, but strictly the bi-directionality should be allowed for at query time. For example, a SPARQL query over the graph in Figure A.1, to list any archive

⁸The SITE-ARC table in Fig. A.1 is based on a real RCAHMS database table, which uses a concatenation of `siteNo` and `arcNo` as its primary key. This is awkward in RDF, where we want a single identifier for the resource node, so an extra column (`siteArcNo`) was added as a preliminary step.

⁹Since all the properties are potentially reversible the directionality is not particularly significant, except as regards the semantics of the statements (see Section A). This link is described as “special” because it can validly appear as `(siteX, siteArc, arcY)` or `(arcY, siteArc, siteX)`, which is not true in general for *Tether*.

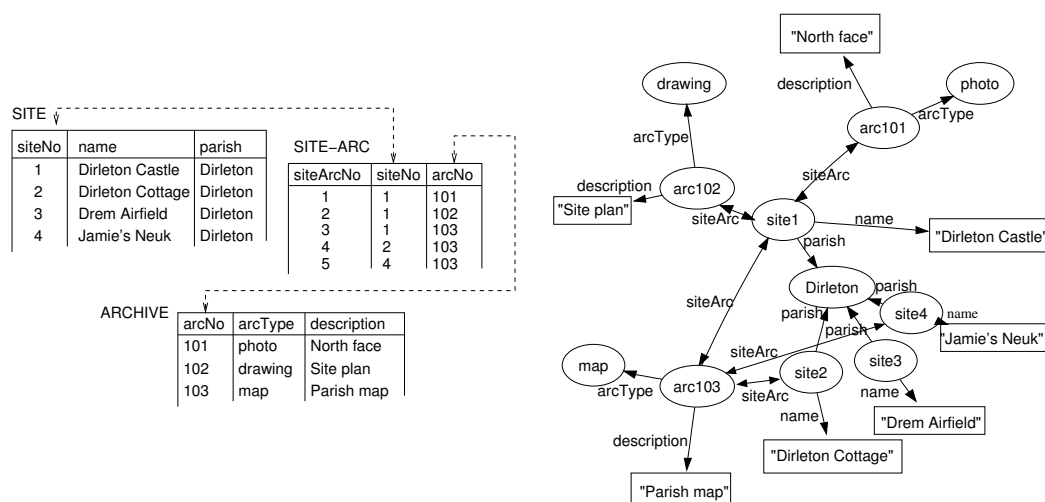


Figure A.1: The table fragments are translated by creating one triple for each column of each row. For the subject of each triple an ID string (*site* or *arc*) is appended to the primary key field. The method takes advantage of the fact that the intermediary table between *SITE* and *ARCHIVE* has no local attributes beyond the necessary foreign keys.

items associated with sites, could be expressed as:

```
SELECT ?sitename ?arcitem ?arcdesc
WHERE {
  ?site name ?sitename .
  OPTIONAL {
    {{?site siteArc ?arc} UNION {?arc siteArc ?site}}
    ?arc arcType ?arcitem .
    ?arc description ?arcdesc . }
}
```

The **UNION** clause ensures that all *siteArc* links are found, whichever way round they point. This query will return the following results:

SITENAME	ARCITEM	ARCDISC
"Dirleton Castle"	photo	"North face"
"Dirleton Castle"	drawing	"Site plan"
"Dirleton Castle"	map	"Parish map"
"Dirleton Cottage"	map	"Parish map"
"Drem Airfield"		
"Jamie's Neuk"	map	"Parish map"

In much the same way, the translation procedure can eliminate an intermediate node where the database uses codes for values taken from a picklist, by pre-joining the

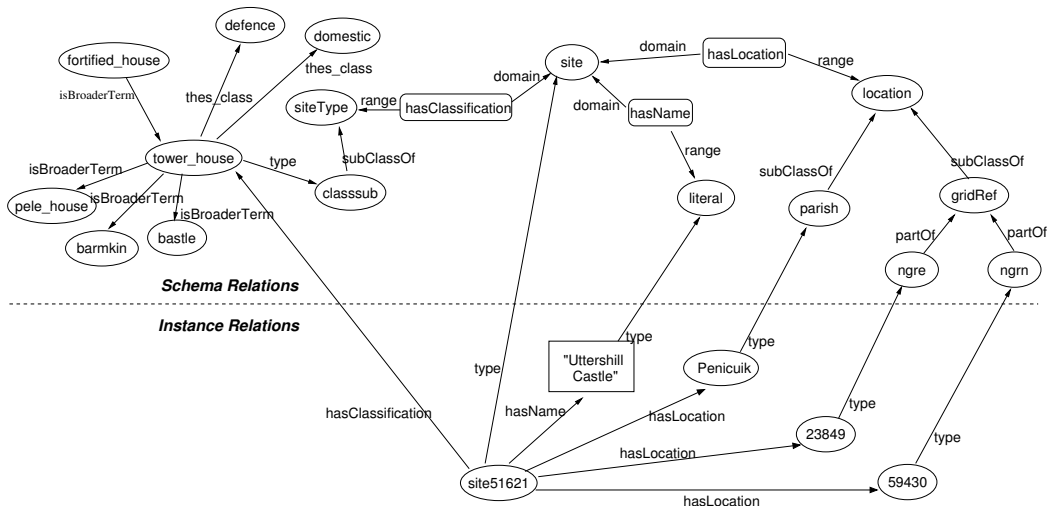


Figure A.2: If the graph schema were hand-designed instead of automatically generated, complexity could be moved into the schema, with simpler instance relations.

relevant tables. For example, `region` is held on site records as a code, pointing at a lookup table of region names. Instead of holding a triple such as (`site1`, `region`, "01") and then having to connect "01" to the region it refers to, we can de-reference the link and produce the triple (`site1`, `region`, `Borders`). In practice it will probably be necessary to hold a literal for `Borders` region and also give it a URI as a resource that can be the subject of other statements, so there will be a number of related triples. The high number of triples required makes it even more desirable to eliminate any that are *not* actually needed.

In order to make the summarisation step (described in Section A) tractable, it will (sadly) be necessary to introduce some manual schema design, rather than using an entirely automatic procedure. As they stand, each of the source databases has hundreds of separate fields, which translate into properties or predicates in RDF — the arcs on the graph. The intention is to simplify the graph by grouping similar predicates together, leaving the details of the relationships between them to the class or schema level as `rdf:type` and `rdfs:subClassOf` properties. So, for example, a new "location" predicate might include database attributes like "parish", "county", "grid reference", "postal address", and so on. Figure A.2 shows an example.

Relation Extraction from Free Text

The final and most interesting step is to populate the graph database with information derived from free text in the database records. This is an active research field and there are many possible approaches. See Riloff and Lorenzen (1999); Schutz and Buitelaar (2005); Huang et al. (2004) and Yangarber and Grishman (2001) for a contrasting range of examples. Most methods use some combination of machine learning tools and hand-crafted rules. For this project methods that can cope efficiently with large data volumes will be preferred, even if this means some sacrifice of thoroughness.

In outline, the steps that will be followed are:

1. Package the text as individual documents, linked to their parent database records. Then carry out standard NLP preprocessing steps such as tokenisation, part of speech tagging and chunking (i.e. identifying verb phrases, noun phrases, etc.)
2. Perform named entity recognition and classification. This is another standard NLP procedure, identifying references to significant “entities” such as people, places, site classifications and so on. The recognition step finds the strings (such as “Charles Rennie Mackintosh”) and the classification step assigns a class label (“person” in this case, or “architect” if the classification is more granular).
3. Deal, as well as possible, with the problems of co-reference resolution (the same entity appearing in multiple surface forms, such as “C R Mackintosh”, “Rennie Mackintosh”), and anaphora (e.g. pronouns will be replaced by their referents). Translate each entity into a canonical form with a URI, as they will become resources in the RDF graph.
4. Identify candidate predicates by doing frequency analysis on verb phrases. Cluster the candidates into synonym groups using a word-to-word distance measure. (There are several approaches to the word distance problem, often using WordNet (Miller et al., 1990). Cilibrasi and Vitanyi (2004) gives an interesting alternative, the “Normalised Google distance”.) If necessary, prune the set of synonym groups to a manageable number. They will become RDF properties.
5. Build triples from the sentences, where the predicate is one of the set just described and the subject or object is a named entity, or both are. If there is only one NE, use the noun phrase or prepositional phrase required by the predicate.

The individual steps of this procedure have been tested with subsets of the data, to check their basic viability. Carrying out the whole process and evaluating it accurately will form a significant portion of the overall project.

Application Design

Once the RDF database has been fully populated, a query application will be built over it. Two significant aspects of the design are discussed in this section.

Querying Graph Data

There are any number of languages available for querying RDF data, and SPARQL¹³ is the preferred choice here. Most of the Semantic Web tools are restricted (so far, at least) to simple subgraph matching and do not support graph searching algorithms, as is discussed by Angles and Gutierrez (2005) and Stuckenschmidt (2005). This seems curious, as one might have assumed that the benefit of translation to graph format was to allow functions like finding shortest paths between nodes, comparing the degree of nodes, and doing k -neighbourhood queries. For this project it seemed preferable to work within established or emerging standards, so some functions must be dispensed with. However, data repositories like Jena, which use a RDBMS back-end store, have to perform a SPARQL-to-SQL translation to send queries to the stored graph; so the possibility of using SQL directly against the triples tables is open. (Degree of node queries, for example, are straightforward in SQL, and k -neighbourhood queries can be done if k is fixed.)

To illustrate the proposed query procedure, consider a practical example such as a user query for “burial customs”, against the RCAHMS data. This is a perfectly reasonable query from a non-specialist, but is too general to produce particularly useful results in practice. Querying CANMORE¹⁴ for `Site Type = "burial"` (the nearest approximation to the query that’s possible) produces 2,588 hits in either alphabetic or geographical order, with no subdivision between, say, 20th Century or Iron Age burial grounds. The *Tether* system will try to provide more guidance and context, by breaking down the mass of results into categories. The steps would be:

1. Check the query terms against the graph database to find preferred terms, related

¹³<http://www.w3.org/TR/rdf-sparql-query/>

¹⁴The current RCAHMS online query facility, available at <http://www.rcahms.gov.uk/>.

terms and subtype terms. For this example, we would get preferred terms like “burial”, “funerary site”, “burial enclosure”, “burial cairn”; non-preferred terms including “boat burial”, “Viking burial” and “burial chamber”; and narrower terms like “bog burial”, “cremation”, “plague burial”, and “chambered cairn”.

2. The term “custom” is not likely to be found in a thesaurus (it does not occur in the Thesaurus of Monument Types), so it would pass on as an unprocessed term. The likeliest place for a match is within parts of the graph database derived from the free text. If no match is found the term would be discarded;¹⁵ otherwise the process skips the next step, finds the “site” nodes related to each match found, and carries on from Step 4.
3. Establish that “burial” is a “site classification”. Find all the nodes of type “site” with “classification” edges linking to one of the terms in the list built at Step 1.
4. Collect the *other* properties of these site nodes, e.g. location, period, associated people or events and so forth. Count the numbers within each broad group of properties (possible in SQL).
5. Amalgamate threads returned by each query term and present the user with a summary of the top groups, that might be of the form:

<i>Period:</i>	Iron Age, 350	Mediaeval, 640	Modern, 820	Others, 550
<i>Location:</i>	Strathclyde, 280	Lothian, 560	Orkney, 730	Others, 990
<i>Has archive?:</i>	Yes, 1500	No, 1100		

 ...etc. (These figures are notional.)
6. Allow the user to choose extra search criteria from these groups, e.g. Modern sites in the Lothians with archive material, and combine these criteria with the original query to produce the final result.

The differences between this and other query interfaces that allow combinations of search criteria are firstly that the criteria are specifically tailored for each query and guaranteed to produce relevant results for it, and secondly that no expertise is assumed on the part of the user, who only has to pick from criteria offered, not specify them.

¹⁵One possibility is to use WordNet to find synonyms for terms that have no match in the graph database.

Presentation of Results

One of the difficulties for organisations that want to make their material available to a wider audience is that it may not be in a suitable form for presentation to the lay reader. A text written for professional archaeologists may be almost unintelligible to, say, a school-child studying Scottish history. Ideally one would like to be able to present information on the same topics in different ways to different types of user, and perhaps even in the reader's native language. Natural language generation techniques make these realistic goals. The *M-PIRO* project (Isard et al., 2003) produced a system for generating descriptions of museum objects in different languages and for different levels of user. The system has been successfully adapted to handle a sample of RCAHMS data, as a proof-of-concept demonstrator. In principle at least, the small hand-built ontology used in the demonstrator could be replaced with a large RDF database. Thus there is the possibility of producing descriptive text output, tailored for the individual user, from *Tether*. This would not be practicable if the data were not held as RDF.

Conclusion

This paper has given an overview of a research project that is currently underway. Although the primary goals are practical ones — concerned with real data accessibility issues — the aim is also to explore Semantic Web and NLP techniques as generic tools, and assess their applicability to cultural heritage information. One of the contentions being tested is that cultural data is, at present, generally held in a format that is not ideal for it, and translating it into a different layout will make it more accessible, particularly to the non-expert user who knows neither the data structure nor the jargon of the field.

The results will be evaluated against standard metrics for recall and precision, and in comparison to what is available in other existing systems. To be counted successful, the system should have performance comparable with existing SQL-based applications, increased retrieval power, and be simpler to use.

Appendix B

Timetable for remaining work

Relation Extraction work	–	by end Oct 2007
Finish database conversion, integrate data	–	by end Nov 2007
Query experiments	–	by end Jan 2008
Finish thesis	–	by end Mar 2008

Bibliography

- Alex, B., Haddow, B., and Grover, C. (2007). Recognising nested named entities in biomedical text. In *Proceedings of BioNLP 2007*, Prague, Czech Republic.
- Alexander, N., Lopez, X., Ravada, S., Stephens, S., and Wang, J. (2005). RDF data model in Oracle. Technical white paper, Oracle Corporation.
- Angles, R. and Gutierrez, C. (2005). Querying RDF data from a graph database perspective. In *2nd. European Semantic Web Conference (ESWC2005)*, volume 3532, pages 346–360, Heraklion, Greece. Lecture Notes in Computer Science.
- Angles, R., Gutierrez, C., and Hayes, J. (2004). RDF query languages need support for graph properties. Technical Report TR/DCC-2004-3, University of Chile, Department of Computer Science.
- Anyanwu, K. and Sheth, A. (2003). ρ -queries: Enabling querying for semantic associations on the Semantic Web. In *Proceedings of the Twelfth International World Wide Web Conference*, pages 690–699, Budapest, Hungary.
- Bailey, R. (2004). Learning and its delivery at RCAHMS. *RCAHMS Annual Review 2003-04*.
- Berners-Lee, T. (2006). Relational Databases on the Semantic Web. Internet note. v 1.22 2006/02/01 (originally published September 1998).
- Byrne, K. (2006). Tethering cultural data with RDF. In *JUC2006 (Jena User Conference 2006)*, Bristol, UK.
- Cilibrasi, R. and Vitanyi, P. M. B. (2004). Automatic meaning discovery using Google. Internet.
- Codd, E. F. and Date, C. J. (1993). Much ado about nothing. *Database Programming & Design*, 6(10).

- Collier, N., Park, H. S., Ogata, N., Tateisi, Y., Nobata, C., Ohta, T., Sekimizu, T., Imai, H., Ibushi, K., and Ichi Tsujii, J. (1999). The GENIA Project: Corpus-based Knowledge Acquisition and Information Extraction from Genome Research Papers. In *Proceedings of EACL'99*, pages 271–272.
- Consens, M. P. and Mendelzon, A. O. (1990). The G+/GraphLog visual query system. In Garcia-Molina, H. and Jagadish, H. V., editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, page 388, Atlantic City, NJ. ACM Press. ISSN:0163-5808.
- Crofts, N., Doerr, M., and Gill, T. (2003). The CIDOC conceptual reference model: A standard for communicating cultural contents. *Cultivate Interactive*, (Issue 9).
- Curran, J. and Clark, S. (2003). Maximum entropy tagging for named entity recognition. Informatics research report, University of Edinburgh, School of Informatics, ICCS.
- Cyganiak, R. (2005). Note on database layouts for SPARQL datastores. Technical Report HPL-2005-171, HP Laboratories, Bristol.
- Date, C. J. (2003). *An Introduction to Database Systems*. Addison-Wesley, 8th edition.
- Gu, B. (2006). Recognizing Nested Named Entities in GENIA corpus. In *Proceedings of the BioNLP Workshop on Linking Natural Language Processing and Biology at HLT-NAACL 06*, pages 112–113, New York City. Association for Computational Linguistics.
- Harris, S. (2005). SPARQL query processing with conventional relational database systems. In *International Workshop on Scalable Semantic Web Knowledge Base System (SSWS 2005)*.
- HP (2004). Jena2 database interface - database layout. Internet.
- Huang, M., Zhu, X., Hao, Y., Payan, D. G., Qu, K., and Li, M. (2004). Discovering patterns to extract protein-protein interactions from full texts. *Bioinformatics*, 20(18):3604–3612.
- Isard, A., Oberlander, J., Matheson, C., and Androutsopoulos, I. (2003). Speaking the users' languages. *IEEE Intelligent Systems*, 18(1):40–45.

- McDonald, R., Crammer, K., and Pereira, F. (2005). Flexible text segmentation with structured multilabel classification. In *Proceedings of EMNLP05*.
- MIDAS (2003). *MIDAS: A Manual and Data Standard for Monument Inventories*. English Heritage, Data Standards Unit, National Monuments Record Centre, Swindon, 3rd reprint edition. ISBN: 1-873592-33-7.
- Mikheev, A., Moens, M., and Grover, C. (1999). Named Entity recognition without gazetteers. In *Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL'99)*, pages 1–8.
- Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. J. (1990). Introduction to WordNet: an on-line lexical database. *International Journal of Lexicography*, 3(4):235–244. Revised August 1993.
- Navigli, R. and Velardi, P. (2003). An analysis of ontology-based query expansion strategies. In *Proceedings of the International Workshop on Adaptive Text Extraction and Mining*.
- Nissim, M. and Krymolowski, Y. (2003). RCAHMS Annotation Guidelines. Held at [/group/ltg/projects/SEER/Annotation/Guidelines/guide.tex](#).
- Ramakrishnan, R. and Gehrke, J. (2000). *Database Management Systems*. McGraw-Hill, 2nd edition.
- Riloff, E. and Lorenzen, J. (1999). Extraction-based text categorization: Generating domain-specific role relationships automatically. In Strzalkowski, T., editor, *Natural Language Information Retrieval*, chapter 7, pages 167–196. Kluwer Academic. strz99.
- Schutz, A. and Buitelaar, P. (2005). *RelExt*: a tool for relation extraction from text in ontology extension. In *Proceedings of the 4th International Semantic Web Conference (ISWC)*, Galway, Ireland.
- Smith, A. and Osborne, M. (2006). Using gazetteers in discriminative information extraction. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 133–140, New York City. Association for Computational Linguistics.

- Stuckenschmidt, H. (2005). Towards an RDF query language - comments on an emerging standard. SIG SEMIS (Semantic Web and Information Systems).
- Tudhope, D. and Binding, C. (2004). A Case Study of a Faceted Approach to Knowledge Organisation and Retrieval in the Cultural Heritage Sector. *Digicult – Thematic Issues*, (6: Resource Discovery Technologies for the Heritage Sector):28–33.
- Turnbull, G. (2003). Aerofilms collection project consultancy: Audience development and access plan. Report by SCRAN for RCAHMS.
- Yangarber, R. and Grishman, R. (2001). Machine learning of extraction patterns from unannotated corpora: Position statement. In *Proceedings of Workshop on Machine Learning for Information Extraction*, pages 76–83, Berlin.
- Zhang, L. and Yao, T. (2003). Filtering Junk Mail with a Maximum Entropy Model. In *Proceedings of 20th International Conference on Computer Processing of Oriental Languages (ICCPOL03)*, pages 446–453.
- Zhou, G., Zhang, J., Su, J., Shen, D., and Tan, C. (2004). Recognizing Names in Biomedical Texts: a Machine Learning Approach. *Bioinformatics*, 20(7):1178–1190.