Information Extraction in the KELP Framework

Robert Dale, Marc Tilbrook
Centre for Language Technology
Macquarie University
E-mail {rdale|marct}@ics.mq.edu.au
Cécile Paris
Intelligent Interactive Technology Research Group
CSIRO Mathematical and Information Sciences
Cecile.Paris@cmis.csiro.au

Abstract

In this paper, we describe some early steps in a new approach to information extraction. The aim of the KELP *project is to combine a variety of natural language processing techniques so that we can extract useful elements of information from a collection of documents and then re-present this information tailored to the needs of a specific user. Our focus here is on how we can build richly structured data objects by extracting information from web pages; as an example, we describe the extraction of information from web pages that describe laptop computers. A principle goal of this work is the separation of different components of the information extraction task so as to increase portability.*

Keywords Information extraction, natural language generation, document personalisation.

1 Introduction

Information Extraction (IE [1]; the process of identifying a pre-specified set of key data elements from a free-text data source) is widely recognised as one of the more successful spin-off technologies to come from the field of natural language processing. The DARPA-funded Message Understanding Conferences resulted in a number of systems that could extract from texts, with reasonable results, specific information about complex events such as terrorist incidents or corporate takeovers. In each case, the task is manageable because (a) some other means has determined that the document being analysed falls within the target domain, and (b) the key information required is typically only a very small subset of the content of the document. A major component task is named entity recognition [3], whereby people, places and organizations are located and tracked in texts; other processing can then take the results of this process to build higher order data structures, establishing, for example, who did what to who and when. In this paper, we describe a new mechanism that relies on a hand-constructed knowledge template, along with a general inference mechanism we call **path merging**, to reconcile and combine the informational elements found in a text.

Proceedings of the 7th Australasian Document Computing Symposium,
Sydney, Australia, December 16, 2002.

2 Background

The goal of the KELP[1] project is to develop technology that can extract information from a collection of web pages that describe similar things, and then collate and re-present this information in such a way as for a user to make it easy to compare those things. Suppose you are interested in purchasing a new cell phone: you could check out a consumer magazine, or visit a web site that presents comparisons of the different available models, but those sources are typically not up-to-date. You might visit the manufacturers' web pages to obtain information from the original sources, but this is a painful, slow process, and comparison is hindered by the different terminology each vendor uses; further, all these sources provide information for a 'typical' visitor, rather than tailoring what they present to your particular needs and interests.

In KELP, we aim to build technology that can mine the information from these source web pages, and then, using techniques we have discussed elsewhere (see [2]), re-present it in a form that is tailored to needs and interests captured in a specific user profile. Our initial experiments have been in the context of web pages that describe laptop computers. Based on an analysis of around 120 web

[1] KELP stands for Knowledge Extraction and Linguistic Presentation, emphasising that the project is concerned both with document analysis and document production techniques. The KELP project is funded by CSIRO, whose support we gratefully acknowledge.

```
<?xml version="1.0" ?>
  <laptop_info>
    <laptop_id>
      <manufacturer>NEC</manufacturer>
      <series>Versa</series>
      <model>Premium PIII 1GHz</model>
    </laptop_id>
  <components>
    <cpu>
      <cpu_type>Pentium III</cpu_type>
      <cpu_speed>
        <number>1</number>
        <unit>GHz</unit>
      </cpu_speed>
    </cpu>
  ...
</laptop_info>
```

Figure 1: A portion of a filled knowledge object

pages describing laptop computers, we developed a template that captures the range of information we might want to extract regarding a laptop computer, and then set about developing techniques to extract the relevant data from these web pages. Part of a typical filled template, or, as we call these structures within KELP, a **knowledge object** or KO, is shown in Figure 1. We have elided this structure to make it fit the space available; it should be obvious that the quantity and complexity of the data to be extracted is significant.

3 Our Approach

Our starting point is a definition of a KO, as shown in the previous section; defined in our current system via an XML DTD, this is a hierarchical structure that specifies the nature of the data to be extracted from the source documents.

The simple key to our approach is to recognize that fragments of the text can be correlated with different parts of the KO structure. For example, we know that the manufacturer will be a company name; and we know that the hard disk capacity will be measured in Mb or Gb. Thus, we can assign type information to the leaf nodes in this structure. At the same time, words in the text can serve as indicators of particular attributes: so, if a sentence contains the phrase *removable storage*, we can use this to hypothesise the presence of a particular attribute in the text. We think of each such text fragment as a piece of evidence; faced with evidence from the text of a collection of attributes and values, the goal is then to combine this information to populate a KO.

The architectural model we use thus consists of the following components. An **annotation resource file** provides a correlation between between arbitrarily complex textual patterns and the knowledge object constituents for which these patterns provide evidence.

A **text scanner** processes each input document, searching for the patterns specified in the annotation resource file. Each time a match is found, this generates a hypothesis, in the form of a **path fragment** associated with a piece of text. The consequence of processing a document is thus a collection of path fragments that capture the evidence found in the document.

The **path combiner** then takes this set of path fragments and attempts to put these together to build a complete knowledge object. Path fragments may contribute to multiple hypotheses about the object being constructed, so the combiner uses the target knowledge object template as a source of constraints on what is possible.

In most situations, this will not produce a single KO as a result; there will still be ambiguities resulting from textual fragments providing evidence for more than one KO constituent. At this point, we resort to a collection of **inference strategies** to resolve the ambiguities.

Of course, if we had full broad-coverage natural language processing available, then this would achieve the same result: we could just parse the entire text, carry out semantic analysis, and build a representation of the text. However, such an approach is not feasible given the current state of NLP technology, so our aim here is to build on the simpler pattern-matching approaches found in the IE literature, augmented with more sophisticated higher-level processing. Our architectural breakdown stratifies the knowledge used in a way that supports easy maintenance and portability: the annotation resource file is a relatively simple declarative knowledge source developed anew for each domain; the text scanner and path combiner are generic components that do not embody any domain-specific knowledge; and higher-level knowledge of the domain is factored out into the KO template and the inference strategies.

3.1 Paths

We can view a KO as a graph structure (and for present purposes a tree) into which extracted information can be placed. The arcs between nodes in this tree are labelled with the same names as the element tags in the XML DTD; a path is a sequence of arcs in the tree. Each attribute corresponds to path from the root of tree, and each value is a data element that resides at the end of that path. Paths may share common initial subsequences: this means that we use hierarchy in the tree to cluster related pieces of information into subtrees.

We use the notation A:B:C to notate **path fragments**. If a path is from the root of the tree, the path contains an initial ':'; if the path has a value at its end, then we use a terminal ':' to indicate this. A **path equation** indicates the value at the end of a path: for example

:laptop:iodevices:keyboard:numkeys: = 131

Each piece of evidence we find in a text can be annotated with a path fragment: this fragment can be of various kinds depending upon how strong the evidence is. The example above is a **complete** path fragment, where there is no doubt that a particular value is the value of a particular attribute.

A path fragment can be **initial**: this means that we don't have a complete path but we do have some initial sequence of arcs in a path. So, the string *on-board memory* might correspond to the following annotation:

:laptop:memory:on-board

We don't know at this point what aspect of the on-board memory is being described.

A path fragment can be **medial**: this means that we don't have a complete path but we do have the some sequence of arcs in the middle of a path. So, a string like *memory capacity (maximum)* may correspond to main memory, graphics memory, or perhaps some other kind of memory; this corresponds to the following medial path fragment:

memory:maximum:bytesize

Finally, a path fragment can be **final**. This means we have some sequence of arcs at the end of a path. So, the string *2Gb* corresponds to the following pair of path fragments:

bytesize:unit: = Gb
bytesize:number: = 2

To operationalise these notions, the annotation resource file correlates arbitrarily complex textual patterns with path fragments. These patterns are then used by the text scanner to generate hypotheses about the information in the text, expressed in terms of the path fragments; the complete analysis of a text thus results in a set of textual clues and their corresponding hypothesised path fragments.

3.2 Path Merging

A KO consists of set of paths, and a fully instantiated KO has a value for each path that makes up the KO. We can characterise an instantiated KO by a set of path equations:

:laptop:model:manufacturer: = Dell
:laptop:model:series: = Inspiron
...
:laptop:iodevices:mouse:numbuttons: = 3

From a text, we derive a set of path fragments like those shown in the previous section. Our goal is to take this collection of path fragments and to derive from them a set of path equations that define an instantiated KO. Formally there are many combinations of path fragments that we could entertain.[2] A number of cases need to be considered.

First, we do not have to do anything to path fragments which are complete; note, however, that we do not want to just forget about these, since their presence may rule out some other possible combinations (in the example above, if we are tempted to merge two path fragments that would give us a different number of keys, then we have a good reason for not doing this).

Second, two path fragments may share some arcs: so, in the above, a possible combination results in

memory:maximum:bytesize:unit: = Gb

This is a possible combination but not a necessary one: arc labels are not unique, so it's possible that this particular bytesize:unit fragment does not belong with the memory:maximum:bytesize fragment.

Third, from a formal point of view, any pair of paths can be combined, with the exception that an initial path can only appear at the front of a combined path, and a terminal path can only appear at the end of a combined path. In such cases, where there is no overlap, there is essentially missing material in the middle, which we indicate using '...'. So, we might have a combined path that looks something like the following':

:laptop:memory:...:bytesize:unit: = Gb

This is a case where we have some possible evidence for a memory size but we don't know if it is the standard on-board memory, the expanded-to-maximum memory size, or some other aspect of memory size. Of course, not all formally possible combined paths are actually possible. The KO definition provides a way of ruling out impossible path combinations. Our technique for filtering the paths is as follows.

First, we take the set of paths that constitute a KO, called the KO **path set**; this will look something like the following:

:laptop:model:manufacturer:
:laptop:model:series:
...
:laptop:iodevices:mouse:numbuttons:
...

Then, we take the set of path fragments derived from the document. We first separate out the medial paths and the complete paths, leaving the initial paths and final paths. We then produce all

[2]The ideas discussed here have been strongly influenced by work in graph-based unification [4].

possible initial × final combinations, resulting in what we call the IF **path set**. Each element of the IF path set has the form

:A:B:C:...:X:Y:Z:

We then compare each element of the IF path set against the KO path set. Any match of an IF path against the KO path set constitutes an **instantiation**: it provides a possible substitution for the '...' part. Note that any IF path may result in multiple instantiations. If an IF path results in no instantiations, then it is not completable given the KO definition, and can be discarded. The remaining IF paths make up the **filtered** IF **path set**; and each element of this set may correspond to multiple instantiations. We notate instantiations as follows:

:A:B:C:[P:Q:R]:X:Y:Z:

This indicates that P:Q:R is a possible completion derived from the KO path set. In effect, material between square brackets is hypothesized on the basis of top-down knowledge; we have not extracted direct evidence for it from the text.

Next we take the medial paths and see if they support any of these instantiations by matching them against the instantiations. Again, a medial path may support multiple instantiations. A medial path may actually overlap with the initial or final path in an instantiation. By combining this information, we produce a set of paths built from the initial, medial and final path fragments in the text, and filtered using the KO path set. We then need to reduce the set of instantiations in the filtered IMF path set so that we end up with a set where each I, M or F element only plays a role once. Some combinations of the contributing I, M and F fragments are are more likely than others. We therefore need to assess each combination, and where there is a competing demand for a piece of evidence, determine which of the alternative uses of that evidence is most plausible.

3.3 Adding Reasoning

The set of hypotheses constructed so far uses a combination of bottom-up knowledge from the textual source itself, and top-down knowledge from the KO. This does not necessarily result in a completely instantiated KO, and so we need to add heuristics that allow us to choose between competing hypotheses.

Note that the architectural separation we have adopted allows the incorporation at this stage of arbitrary intelligence to the process, thus focussing the knowledge-based processing in one place; here, we describe the incorporation of a simple heuristic based on a distance metric.

Ultimately, where we have competing instantiations, we can assign probabilities to each. One simple probability measure is based on the distance between the initial path fragment (which generally corresponds to the attribute being considered) and the final path fragment (which corresponds to the value being assigned to that attribute): we can assign scores so that those instantiations that have closer I and F evidence will score higher. Then, we select from this set a smaller set of paths that (a) uses each piece of evidence only once and (b) takes account of the scores. The result is a set of hypotheses that populate the KO using all the data located in the source document, with each piece of evidence being used once.

This is, of course, a very simple technique, but one that seems to work well on the basis of our initial experiments, at least for information extracted from free-form text. Far more elaborate heuristics could be incorporated in the same way.

4 Conclusions

We have presented an approach to information extraction that separates out the different knowledge sources and types of knowledge required. The approach makes use of both top-down and bottom-up knowledge sources, and neatly partitions domain-specific and domain-independent processing: we believe the mechanisms described here should be easily portable to a new domain by constructing a knowledge object template for that domain, and an appropriate annotation resource file. The text scanner and path combining modules are domain independent, as are the current inference strategies; as the work develops, we would expect the inference strategies to break down into those which are domain-dependent and those which are domain-independent.

References

[1] J. Cowie and W. Lehnert. Information extraction. *Communications of the ACM*, Volume 39, Number 1, pages 80–91, 1996.

[2] R Dale, S J Green, M Milosavljevic, C Paris, C Verspoor and S Williams. Using natural language generation techniques to produce virtual documents. In *Proceedings of the Third Australian Document Computing Symposium (ADCS'98)*, Sydney, Australia, August 21 1998.

[3] Andrei Mikheev, Claire Grover and Marc Moens. XML tools and architecture for named entity recognition. *Markup Languages*, Volume 1, Number 3, pages 89–113, 1999.

[4] S. Shieber. *An Introduction to Unification-Based Approaches to Grammar.* CSLI Lecture Notes. Chicago University Press, Chicago, 1986.