Automatic Evaluation and Composition of NLP Pipelines with Web Services

Harry Halpin*

*Institute for Communicating and Collaborative Systems
University of Edinburgh & 2 Buccleuch Place EH8 9LW Edinburgh, UK
H.Halpin@ed.ac.uk

Abstract

We describe the innovative use of describing an existing natural language "pipeline" using the Semantic Web, and focus on how the performance and results of the components may be described. Earlier work has shown how NLP Web Services can be automatically composed via Semantic Web Service composition, and once the results of NLP components can be stored directly, they can also be used to direct the composition, leading to advances in the sharing and evaluation of NLP resources.

1. Introduction

Taking advantage of the convergence of Web and natural language technologies, it is now possible to automatically distribute and evaluate multi-component natural language applications using a framework based on Web Services and the Semantic Web. This approach addresses a number of issues with the distribution of natural language processing applications and the difficulty of evaluating them, even when well-known evaluation metrics and corpora exist (Leidner, 2003). Bundling natural language processing components as Web Services allows them to be accessible in multiple locations (Grover et al., 2004). Also, an extensible framework based on using Semantic Web ontologies to describe the components makes the automated composition of natural language applications possible (Klein and Potter, 2004). One area that has not yet been explored in the use of Semantic Web technologies for recording the automatic evaluation of NLP components, and a global overview and a brief proposal is presented here.

Our driving example is that of an existing pipeline of natural language components used to analyze students plots in the *story re-writing task* (Halpin et al., 2004). This pipeline has been shown successfully in the past to analyze and help students writing, but due to the lack of access to powerful enough computers at schools in rural Scotland, only access to the program via the Web is currently feasible. This application is primarily a multi-component pipeline that uses automatically extracted predicate-argument structures to evaluate the text. Note there are a wide variety of machine-learners, chunkers, and taggers that can be used in this application, and all of these and their interactions effect our final results. Also, like in many applications, these components fulfill *functional* roles that can be fulfilled by any number of components, and everything else being equal, the selection criteria for each component is usually its performance on various corpora of interest. However, the performance of each component also depends on the performance of the components preceding it in the pipeline. Therefore, final evaluation needs to take into account not only the final results of the last component, but the results of preceding components in the application.

2. Using the Semantic Web

We present a framework that enables NLP components as Web Services, and use the Semantic Web to describe corpora, results and experiments. The central point of Web Services is that each program can be given a URI (Uniform Resource Identifier). This URI allows access to the program from over the Web using SOAP or REST architecture (Narayanan and McIlrath, 2002). In comparison, if Web Services gives URIs to programs such as taggers, then the Semantic Web gives URIs to machine-readable data, such as corpora. The Semantic Web also allows one to construct statements about these URIs, where each atom in the statement is also given a URI. One can then describe NLP experiments by "linking" together corpora, applications, and experimental results via the Semantic Web. When the URI for the vocabulary is resolved, it can provide a human-readable definition of the vocabulary. Unlike standards that are essentially "closed" to revision and that may not be capable of expressing the particularities of a given experiment, the Semantic Web is easily extensible in a decentralized manner: to create a new RDF vocabulary one just mints a new URI and hopefully documents it.

Semantic Web "links" can be given as RDF statements (Resource Description Framework), the foundational Semantic Web standard (Klyne and Carroll, 2004). RDF statements follow the "triple" format and are composed in this manner using N3 syntax:[1] `ns:subject ns:predicate ns:object`. The `ns` signals a *namespace*, such that the local name resolves to the URI given by the namespace by the concatenation of the local name to the namespace. The Semantic Web and Web Services share the same scope: the space of URIs. Also note that a period ends a RDF statement, and a semicolon signals that the subject of a predicate is to repeated in the next statement.

Semantic Web vocabularies have an important strength and limitation: everything must be described as a triple. For example, the namespace `semStories` may denote the "Semantic Stories" corpus of children's stories collected in Scotland. A collection of components that have been enabled as Semantic Web Services are given using the `shallowSem` name, emphasizing the facts that these components have been used to extract "shallow semantics" from text. Therefore, a basic experiment that runs a single NLP component on a single corpus could be construed in triples as $C_1\ A_1\ R_1$ where C_1 is the URI of the corpus, A_1

[1] For a complete description of N3 syntax see `http://www.w3.org/2000/10/swap/Primer.html`

is the URI of the NLP application, and R_1 is the URI of the results. This could be exemplified as:

```
semStories:corpus
shallowSem:components/maxentTagger
semStories:exps/1.
```

If the experiment is needs to be repeated using a different tagger such as the CandC tagger (Clark and Curran, 2004), then the experiment can be phrased as C_1 A_2 R_2, which can be exemplified by:

```
semStory:corpus
shallowSem:components/candcTagger
semStories:exps/2.
```

If one wishes to state more about an experiment, such as explaining what component tokenized the corpus, one could create an additional statement:

```
semStories:corpus
wsNLP:component
shallowSem:components/lxtransduce/token2XML.
```

If the statements were merged it would be unclear if `token2XML` belonged to experiment one (`exps/2`), experiment two (`exps/2`), or both.
This problem is the result of a lack of provenance information. We would like to to group RDF statements (graphs) by experiment. We could use RDF reification, but that would make it nearly impossible to do reasoning, since the use of reification is analogous to jumping to higher-order logic. Reification also is not implemented in current reasoners. A more straightforward approach is to use the widely implemented *named graph* approach, in which a "triple" is given some context by adding an additional URI (the name) to the entire graph (Carroll et al., 2005). Each experiment could be used as the name URI of a named graph, therefore avoiding the "semantic muddle" of the earlier example. So our example could be disambiguated to state the we only know that the `token2XML` program was used in the second experiment simply by using `semStories:exps/2` as the name and qualifying our statement as a named graph.

```
semStories:exps/2 {
semStories:corpus
wsNLP:component
shallowSem:components/lxtransduce/token2XML.
}
```

To discover more information about the experiment one could then retrieve more RDF statements about the experiment (such as the date the experiment took place, the experimenter, and so on) by accessing the URI. We can still sensibly merge URIs in the named graph with statements outside the named graph.

3. The Case for Web Services in NLP

Many natural language applications are composed of a number of differing components, ranging from those involved in text-processing such as tokenizers, chunkers, and part-of-speech taggers to those involved with machine-learning. These components are often assembled using the "pipeline" model, in which the output of one component is the input of another, often with the use of "shims" to convert from one format to another when moving the data between components. In general these applications are "brittle" as regards their implementation since it usually requires considerable effort to make the components work on different platforms, as often the source code and even documentation for the components is unavailable (Leidner, 2003). This is problematic for NLP. Unlike other sciences, NLP experiments usually cannot be replicated if the software and corpora are not available. The experimental design given in many papers cannot tell researchers exactly what components or parameters are responsible for the performance gains due to space constraints. Lastly, the level of domain independence of a particular NLP application can not be evaluated if the application can not be tested on a wide range of different corpora. These factors lead to a situation where research can consist of training and testing machine-learning techniques, dependent on a number of components whose performance is not assessed, over some corpus that is not widely available, with results that can not be reproduced or tested for domain independence. More could be done to allow researchers in NLP to use each others applications, share corpora, and share results.

The problem of distribution and access of NLP components can be solved by Web Services, since it would allow any user to use a SOAP client (or just *http*, in the case of a REST Web Service) to invoke the component by its URI (Grover et al., 2004). This offers the user more freedom than being constrained by one locally installed framework. The owner of the Web Service keeps the installation of the NLP component on their local Web Service-enabled server. Although free access to the source code is desirable, this aspect of Web Services can resolve issues of copyright by allowing the researcher not to release the source code and still let the research community access the NLP component in question. Since the vast majority of components cannot be assumed Web Service-enabled or even use XML in their input and output, there is considerable effort involved in packaging components as Web Services.

Unlike many other frameworks, a Web Services and Semantic Web-based framework provides not only a way of describing NLP applications and experiments, but a method of accessing these applications and running these experiments. To repeat, if a URI denotes a NLP Web Service, one can invoke the service to actually do the processing. If a URI denotes a corpus or experimental results, we can either directly provide the results or corpus from the URI, or in the case of corpus or results with a restricted licenses, provide a redirect to a web page with the conditions of the license. Since the vast majority of components cannot be assumed to be Web Service-enabled or even to use XML in their input and output, there is considerable cost involved in packaging components as Web Services. Web Services that are described using the Semantic Web are Semantic Web Services, and there is much work on using the Semantic Web to automatically compose Semantic Web Services

(Narayanan and McIlrath, 2002).

In order for the tools to interoperate together and have their input and output validated, particular NLP Web Services also must operate in a common XML Schema. While a number of standards exists to provide global standards for various linguistic tasks, we find it more likely that components will each have their own component-specific schemas (Ide and Romary, 2004). If the component schemas are modularized, XML Schema inclusion can be used to construct federated NLP application-specific schemas from the schemas of their respective NLP components. In the case that the schemas overlap or conflict, XSL transformations between output formats and schemas can be provided.

4. Automatic Pipeline Composition

There is rarely only one component in an NLP application. When more than one component is needed and the steps need to be arranged in a simple linear pipeline (or "workflow") of Web Services and if the services (NLP components) needed can be described using the Semantic Web, the pipeline can be composed and executed automatically via a Semantic Web Service composition language such as OWL-S (Klein and Potter, 2004). Although it requires considerable effort to describe Web Service NLP applications using OWL-S (or an alternative methodology), once this work has been done once, it would enable others to use these components in their own experiments with ease. Since the description can be at a level of abstraction above a particular instance of an application, Semantic Web Service composition would allow someone to run applications based on their abstract specification, such as "Coreference Resolution" as opposed to a concrete instantiation of a coreference resolver such as CogNIAC (Baldwin, 1997). This would allow a reasoner to compose an application based on an abstract specification using backward chaining reasoning using a database of components.

To extend our first example, the pipeline may be composed of many different processes operating over a corpus. Using Semantic Web Service composition, if we desire to create predicate-argument structures based on verbs and nouns from chunks, we can use backward-chaining and reason that first the corpus must be tokenized, then tagged, then chunked. If each of these have been described as a Semantic Web Service, then we can invoke them in order to "compose" our NLP application. Following this example, we modify and extend the ontology given by Klein and Potter to keep track of "shims," schemas, and other useful information in Figure 1 (Klein and Potter, 2004). We describe the RDF in N3 again for readability and skip namespace definitions assuming a default namespace of `wsNLP`. Note the `wsNLP` RDF vocabulary is interoperable with Dublin Core, and many components are sub-classes of Dublin Core, such as `wsNLP:InputFormat` being a subclass of `dc:format`.

5. Describing Corpora using the Semantic Web

We can describe corpora using the Semantic Web to track the transformations of corpora and what components were responsible for each transformation. We can also build off

```
shallowSem:components/SCOLtuples
   dc:title "SCOL Tuples";
   :location shallowSem:components/SCOLtuples;
   :inputType types/chunkedDocument;
   :inputFormat "text/xml";
   :inputSchema shallowSem:schemas/chunkSchema/chunkSchema.xsd;
   :componentType shallowSem:NLPropositionExtractor;
   :outputFormat "rdf/xml";
   :outputSchema shallowSem:schemas/rdfProp/rdfProp.rdfs;
   :outputType types/Proposition;
   dc:creator "Steve Abney";
   dc:description abney:scol1k.tgz;
   :implementationLocation abney:index.html;
   :implementation/sourceLanguage "C";
   :implementation/location Abney:scol1k.tgz;
   :webServiceLocation shallowSem:components/tuples;
   :XMLSchema shallowSem:components/tuples/schema;
   :inputShim shallowSem:components/SCOLtupes/XMLShimInput;
   :outputShim shallowSem:components/SCOLtupes/XMLShimOutput.
   shallowSem:components/SCOLtupes/XMLShimInput
         :inputType :types/chunkedDocument;
         :inputFormat "text/xml";
         :inputSchema shallowSem:schemas/chunkSchema/chunkSchema.xsd;
         :location shallowSem:components/tuples/shim/inputXML;
         :outputType :types/chunkedDocument;
         :outputFormat "text";
         :outputSchema shallowSem:extSchema/breakdelimited.
   shallowSem:components/SCOLtupes/XMLShimOutput
         :inputType :types/chunkedDocument;
         :inputFormat "text";
         :inputSchema shallowSem:textSchema/breaktabdelimited;
         :location shallowSem:components/tuples/shim/outputXML;
         :outputType :types/chunkedDocument;
         :outputFormat  "rdf/xml";
         :outputSchema shallowSem:schemas/SCOLtuples/scolchunk.rdfs.
```

Figure 1: NLP Component Example

```
semStories:corpus
   :version 1.1;
   :corpusLocation semStories:corpus;
   :dateCreated 2002.06.11T1:37:07;
   :documentation semStories:index.html;
   :documentGenre :corpus/types/childrenStories;
   :processStep semStories:corpus#1;
   :processStep semStories:corpus#2;
   :processStep semStories:corpus#3;
   :processStep semStories:corpus#4.
   semStories:corpus#1
      :dateCreated 2003.07.13T4:12:55;
      :documentFormat  "text";
      :corpusLocation semStories:corpus/text.
   semStories:corpus#2
      :dateProcessed 2005.09.22T15:39:22;
      :component shallowSem:components/lxtransduce/token2XML;
      :componentType :types/components/Tokenizer;
      :corpusLocation semStories:corpus/tokenized.
   semStories:corpus#3
      :dateProcessed 2005.09.22T18:40:30;
      :component shallowSem:components/candcTagger;
      :componentType :types/components/POStagged;
      :corpusLocation semStories:corpus/tagged.
   semStories:corpus#4
      :dateProcessed 2005.09.22T18:42:09;
      :component shallowSem:components/glenCova;
      :componentType :types/components/PronounResolver.
```

Figure 2: NLP Corpus Example

of RDF vocabularies such as an Dublin Core and possible mappings of the IMDI standard to RDF.[2] When processing a corpus, if a copy of the corpus or the stand-off annotation already exists in a state needed by the pipeline, instead of re-processing the corpus, one can just use the corpus in the processed state in a pipeline. This prevents corpus processing and component deployment from being done unnecessarily.

We can also reason about what types of processing needs to be done to a corpus in order for it to be used in a pipeline. For example, we may want to tag a particular corpus, but we can not tag a corpus unless it has been tokenized, and so if the corpus does not exist in a tokenized state a tokenizing component can be deployed before the tagger, even if this step has been not made explicit by the invoker. The adding of part-of-speech tags and chunks to our corpus to prepare it for predicate-argument extraction is described in Figure 2.

[2] See `http://www.mpi.nl/imdi` for information.

```
semStories:exps/3
   dc:title  "Plot Rating Classification Experiment";
   :dateRanBegin 2005.09.23T03:01:21;
   :dateRanEnd 2005.09.23T04:25:55;
   :experimenter "Harry Halpin";
   :experimenterContact  "H.Halpin@ed.ac.uk";
   :experimentType :experiments/classification;
   :corpus semStories:corpus;
   :application semStories:application/pipeline;
   :results semStories:results#3.
   semStories:results
      :correctClassified 55.625;
      :incorrectClassified 44.375.
   semStories:exps/3 {
   semStories:application/pipeline
      :processStep semStories:corpus#1;
      :processStep semStories:corpus#2;
      :processStep semStories:corpus#3;
      :processStep semStories:corpus#4;
      :processStep semStories:corpus#5;
      :processStep semStories:corpus#6;
      :processStep semStories:corpus#7;
      :processStep semStories:corpus#8.
   semStories:corpus#1
      :corpusLocation semStories:corpus/text.
   semStories:corpus#2
      :component shallowSem:components/lxtransduce/token2XML;
      :corpusLocation semStories:corpus/tokenized.
   semStories:corpus#3
      :component shallowSem:components/candcTagger;
      :parameter shallowSem:components/candcTagger/model/MUC;
      :corpusLocation semStories:corpus/tagged.
   semStories:corpus#4
      :component shallowSem:components/glenCova.
   semStories:corpus#5
      :component shallowSem:components/SCOLchunk.
   semStories:corpus#6
      :component shallowSem:components/SCOLtuples;
      :corpusLocation semStories:corpus/tuples.
   semStories:corpus#7
      :component shallowSem:components/EventComparison.
   semStories:corpus#8
      :component shallowSem:components/weka/NaiveBayes;
      :parameter shallowSem:components/weka/NaiveBayes/useKernel;
      :results semStories:results#3.
   }
```

Figure 3: NLP Results Example

6. Evaluation-Guided Composition

NLP applications are often difficult to fairly evaluate since the unavailability of applications make it hard to repeat the experiment using different corpora and parameters (Leidner, 2003). Due to space constraints, usually only the best results are presented. It would be useful to be able to save all results in order to prevent unnecessary duplications of experiments. Most importantly, since the results of a NLP application are dependent on particular components and parameters, it is usually through hill-climbing these parameters are optimized, and even then due to the problem of local minima or maxima it is difficult to determine what the actual optimal results are. Therefore, to explore the full space of NLP pipelines and parameters, the results of the NLP experiment themselves should be described using Semantic Web technologies. We present RDF for describing the results of NLP applications with an example shown in Figure 3.

This information can then be used to optimize the composition of NLP web-based applications by considering the composition to be goal-directed by using forward chaining reasoning with Semantic Web Services. In our example, we could aim to minimize the number of instances incorrectly classified, as given by the object of the predicate `incorrectClassified` in the story classification pipeline. In this manner the search-space of possible components can be automatically searched by giving the evaluation metric to be minimized (or maximized) as the goal of the composition. As more and more components, parameters, and corpora in various states of processing are added, the space over which the reasoning must work grows, but so does the likelihood of gaining optimal results.

7. Acknowledgements

Many of these ideas have been worked on in conjunction with Ewan Klein and Henry S. Thompson, often using as a test-bed of components the components of the *LT-XML* and *LX* tools developed by Claire Grover and Richard Tobin. The story re-writing experiment was developed with Johanna Moore (Halpin et al., 2004).

8. References

Breck Baldwin. 1997. CogNIAC : A High Precision Pronoun Resolution Engine.

Jeremy Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. 2005. Named graphs. *Journal of Web Semantics*, 3(4).

Stephen Clark and James Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL 2004)*. Barcelona, Spain.

Claire Grover, Harry Halpin, Ewan Klein, Jochen Leidner, Sebastian Riedel, Sally Scrutchin, and Richard Tobin. 2004. A framework for text mining service. In *Proceedings of the Third UK e-Science Programme All Hands Meeting*. Nottingham, UK.

Harry Halpin, Johanna Moore, and Judy Robertson. 2004. Automatic analysis of plot for story rewriting. In *Proceedings of Empirical Methods in Natural Language Processing*. Barcelona, Spain.

Nancy Ide and Laurent Romary. 2004. International standard for a linguistic annotation framework. *Journal of Natural Language Engineering*, 10:211–225.

Ewan Klein and Stephen Potter. 2004. An ontology for nlp services. In *Proceedings of LREC Workshop on a Registry of Linguistic Data Categories within an Integrated Language Resource Repository Area*.

Graham Klyne and Jeremy Carroll. 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recomendation. http://www.w3.org/TR/rdf-concepts/.

Jochen Leidner. 2003. Current issues in software engineering for natural language processing. In *Proceedings of HLT-NAACL Workshopon Software Engineering and Architecture of Language Technology Systems*. Alberta, Canada.

Srini Narayanan and Sheila McIlrath. 2002. Simulation, verification and automated composition of web services. In *Proceedings of the World Wide Web Conference*, Honolulu, USA.