An Ontology for NLP Services

Ewan Klein, Stephen Potter

National eScience Centre/School of Informatics
University of Edinburgh, Edinburgh, Scotland
{ewan,stephenp}@inf.ed.ac.uk

Abstract

1. Introduction

The main focus of this paper is a framework for describing and discovering NLP processing resources. In many ways, the most difficult aspect of this task is the huge space of options. Even disregarding the wide variety of theoretical models for describing natural languages, and even if we restrict attention exclusively to NLP tools, there is sufficient diversity within the NLP community to provoke much disagreement about the best way to describe such tools. In this paper, we try to narrow down the range of choices by focusing on the following issues. First, we emphasize the role of description in supporting **tool interoperability**. Second, we place interoperability within the context of **service composition**. Third, we develop an ontology of NLP services that is informed by the OWL-S semantic framework (OWL-S).

2. Service Use Cases

2.1. Describing NLP Resources

Before embarking on proposals for how language resources should be described, it is important to consider what requirements need be met. Consequently, we describe two use cases which will guide our design objectives. We are currently interested in addressing the needs of two, rather different, communities of users: **NLP researchers**, on the one hand, and **users of domain-specific text mining services**, on the other.

2.2. The NLP Workbench

NLP researchers frequently wish to construct application-specific systems that combine a variety of tools, some of them in-house and some of them third-party. For example, to carry out a named entity recognition task, Janet might want to run a statistical classifier over a corpus that has been tokenized, tagged with parts of speech (POS) and chunked. She might use her own tokenizer and chunker but use someone else's tagger — say the `TnT` tagger (Brants, 2000). She will almost certainly have to write some glue code in a scripting language to plug these different tools together.

Suppose now that John needs to work with Janet's system a few months later. He wants to try the same experiment, but using a different tagger – say the `CandC` tagger (Curran and Clark, 2003). A number of issues arise. First, how likely is it that John can simply re-run Janet's system as it was? Can he be sure that he's using the same versions of the software, trained on the same data? Can he be sure that he's getting the same results against the same Gold Standard test data? For such a scenario, it would be useful to have some method for recording and archiving particular configurations of tools, together with a record of the results. Moreover, the configuration needs to include information about the location of the relevant versions of the tools. Rerunning the experiment should ideally be as simple as firing up a configuration manager and reloading the configuration script from a pull-down menu.

Assuming that all went smoothly, how likely is it that John can simply remove the call to `TnT` in Janet's script and splice in `CandC` instead? Do they have the same formatting requirements on their input and output? Do the available pre-trained versions of these taggers use the same tagsets? Do they require the same number of command-line arguments? Unfortunately, the answer to these three questions is No.[1] This problem is hard for the human to deal with; consider how much harder it would be to develop a workbench that would automatically check whether two tools could be serially composed. A crucial obstacle to automatic composition is that we lack a general framework for describing NLP tools in terms of their inputs and outputs.

2.3. Text Mining for e-Science

There is increasing interest in deploying text mining tools to assist scientists in various tasks. One example is knowledge discovery in the biomedical domain: a molecular biologist might have a list of 100 genes which were detected in a micro-array experiment and wishes to trawl through the existing published research for papers which throw light on the function of these genes. Another example is the astronomer who detects an X-ray signal in some region of the sky, and wants to investigate the online literature to see if any interesting infra-red signals were detected in the same region. These brief scenarios are special cases of a more general interest in using computing technologies to support scientific research, so-called "e-Science" (Hey and

We are grateful to Steven Bird, Denise Ecklund, Harry Halpin and Jochen Leidner for helpful discussion and comments.

[1] For example, in the case of formatting requirements, `TnT` uses multiple tabs as a separator between word and tag, while by default, `CandC` uses the underscore as a separator; it can, however, be configured to use a single tab as separator.

Trefethen, 2002).

In such cases, we expect that the researcher will be using a *workflow tool*; that is, something that "orchestrates e-Science services so that they co-operate to implement the desired behaviour of the system".[2] In this context, we would want the researcher to have access to a variety of text mining services that will carry out particular tasks within a larger application. A service might be essentially a document classification tool which retrieves documents in response to a string of keywords. However, it might be a more elaborate information extraction system which, say, populates a database that is then queried by some further service.

Each text mining tool which is accessible to the scientist end-user must be able to describe what kind of service it offers, so that it can be discovered by the workflow tool. We would expect there to be a more coarse-grained functionality in this use case: the scientist is unlikely to care which tagger is being used. Nevertheless, it will not always be easy to predict in advance where the external boundary of a text mining service will lie, so in principle the challenge of developing explicit and well-understood interfaces for text mining services overlaps with the previous use case. An additional constraint is that the NLP tools must be interoperable with other services provided by the e-Science workflow environment, and must be accompanied by descriptions which are intelligible to non-NLP practitioners.

3. Design Influences and Goals

In order to tease out requirements, let's reflect further on our first use case. Assume that we are given a simple pipeline of processors which carries out some well-defined text processing task. We wish to remove one processor, say a POS tagger, and splice in a new one, while ensuring that we preserve the overall functionality of the pipeline. This means that we need to abstract away from particular taggers to a class of such tools, all of which carry out the same transformation on their input. At this level, we can talk broadly about **interchangeability** of functionally equivalent processors. On the other side of the coin, information about the input and output parameters of two taggers *A* and *B* must be detailed enough for us to tell whether, when *A* is replaced by *B*, *B* will accepts input from the immediately preceding processor and produce output that is acceptable for the immediately following processor. In other words, we require a processor to be accompanied by metadata which enables us to make decisions about **interoperability**.

de Roure and Hendler (2004) argue that interoperability is a key notion for the e-Science research programme, and that technologies from both the Grid (Foster et al., 2001) and the Semantic Web will underpin the programme. The integration of the two technologies has been dubbed the Semantic Grid and both approaches interoperability as being achieved through deployment of *services*. Foster et al. (2002) give a general characterization of services as "network enabled entities that provide some capability through the exchange of messages", and argue that a service-oriented perspective supports virtualization in which resources can be accessed uniformly despite being being implemented in diverse ways on diverse platforms.

We would like, then, an environment that offers an infrastructure for the discovery, orchestration and invocation of services, and one that is flexible and permits a high degree of re-use and automation of workflows. This desire coincides with the aims of much of the effort in the Semantic Web Services community, and so it is to this community that we look for guidance.

The Semantic Web 'vision' is one of enhancing the representation of information on the web with the addition of well-defined and machine-processable semantics (Berners-Lee et al., 2001), thereby encouraging a greater degree of 'intelligent' automation in interactions with this complex and vast environment. One thread of this initiative concerns the provision of web-based services: the web has great potential as a medium for, on the one hand, web service-providers to advertise their services and conduct their business, and on the other, for those with particular service needs to publicise these needs to the environment so as to have them satisfied.

3.1. Description Logics and OWL-S

A number of *de facto* standards exist for locating and invoking web services; these include Unversal Description, Discovery and Integration protocol (UDDI; Bellwood et al., 2002), a protocol for building and using registries of services, the Web Services Description Language (WSDL Christensen et al., 2001), an XML-based language for describing the operations a service offers, and SOAP (Gudgin et al., 2003), an XML-based messaging protocol for communicating with a service. At the time of writing, however, the discovery and use of the relatively few services which exist relies to a large extent on syntactic matching of terms and on human engineering of the content of the invocation calls to them. In order to move towards a semantic service environment, efforts have been made over the last couple of years to develop the OWL-S (previously DAML-S) upper ontology for describing web services. The intention of this initiative is to provide an XML-based ontology which stipulates the basic information that services should expose to the environment in order to facilitate their automatic discovery, invocation, composition and monitoring (OWL-S). This ontology is specified in the OWL Web Ontology Language which provides a language for specifying Description Logic constructs in the syntax of XML and building on top of the RDF data model. Description Logics (e.g. Baader et al., 2003) form a subset of first-order logics which are particularly suited to the description of hierarchical ontologies of concepts, and possess appealing tractability characteristics. Hence, an OWL document describes a machine-

[2] See `http://www.nesc.ac.uk/esi/events/303/`, which offers a fairly recent overview of current work in the workflow area.

processable ontology or fragment of an ontology.

3.2. OWL-S: Profile, Process and Grounding

The OWL-S ontology is divided into three principal areas (cf. Figure 1). The **Service Profile** is used to describe the purpose of the service, and so primarily has a role in the initial discovery of candidate services for a particular task. For the purposes of this paper, we will concentrate on the use and description of profiles, and hence on NLP service discovery. The **Service Model** describes how the service is performed, and is intended for more detailed consideration of the adequacy of the service for the task, to allow the precise composition and coordination of several services and to enable the execution of the service to be monitored. Finally, the **Service Grounding** specifies in concrete terms how the service is actually invoked, the nature of the messages it expects, the address of the machine and port to which these messages should be addressed and so on. We assume that, in general, if a service profile meets the requirements of a client, then any grounding of that service will be an adequate instantiation.

The role of the Profile, then, is to describe the essential capability of the service by characterizing it in functional terms (in addition, non-functional aspects of the service can be specified through additional 'service parameters'). This functional characterization is expressed by detailing the inputs a service expects, the outputs it produces, the preconditions that are placed on the service and the effects that the service has. As well as characterizing services, the Profile has an additional use: to allow potential clients to specify and query for their desired services (which may be partial or more general in nature where details are irrelevant to the client).

Through the use of these *IOPE (Input-Output-Preconditions-Effects)* parameters, a service (or query) may be described in terms of a transformation of its input data into its output data (for example, a POS tagging service can be described as transforming a document into a tagged document). By 'typing' data in this fashion, we gain the ability to define and instantiate 'semantic pipelines' of data through a workflow consisting of a number of distinct services.

However, another mode of use is possible: by extending the core OWL-S ontology within a particular domain with subclasses of the **Profile** class, we also gain the ability to advertise and request services in terms of their categorization; so one might ask for, say, an **NL-Tagger** if one knew that a tagger was required at this point in the workflow. Both the 'transformation' and 'categorization' modes have their uses, and so it is desirable that they be supported in any environment.

This leads to consideration of precisely how particular services in a particular domain are to be described. The OWL-S ontology is (necessarily) domain-independent: to express concepts of particular domains one has to extend the OWL-S ontology through the introduction and use of additional ontological knowledge. However, the use of domain-specific ontologies in this manner places certain obligations on agents in this domain. For service discovery to be possible, both the service providers and potential clients must use the same ontologies: the former to advertise their services, the latter to formulate their requests.[3] Accordingly, there is a need for a standardization effort within domains in order to develop useful and useable ontological descriptions of services. Section 4. describes one such extension of the OWL-S Profile, for describing NLP services, and in such a manner as to permit both the transformation and categorization modes of use described above.

3.3. Reasoning with Profiles: Brokering

Another implication of our approach is that there is at least one 'broker' agent in the domain that acts as a repository for service advertisements and is able to answer service requests.[4] The locations of these brokers would of necessity be known *a priori* to agents in the domain.

Among the fundamental reasoning capabilities of Description Logics are the subsumption of class terms and the classification of individuals into their appropriate categories or classes. Brokers can exploit these abilities to perform service discovery in a number of different ways. For example, on its advertisement, the profile description can be used to classify this service instance into its appropriate location in the domain ontology. Subsequent queries can be interpreted as defining a class description of the desired services; the instances of classes in the service hierarchy which are equivalent to or subsumed by this class are considered to satisfy this query.[5]

It is with this sort of reasoning in mind that we approach the formalization of the NLP domain.

4. A Profile Hierarchy for Linguistic Resources

If we view NL resources as classes arranged in a hierarchy, then a number of taxonomies are possible. It seems rela-

[3] Alternatively, one could envisage the use of different ontologies, along with descriptions of equivalence mappings between their entities, but this introduces additional engineering and processing overheads. The automation of ontology mapping is a difficult problem, for which there are currently no general solutions.

[4] Different types of broker are possible. The simplest (sometimes termed a 'matchmaker' agent) would return matching advertisements to the requesting agent, which is then responsible for selecting and invoking one of these services. More sophisticated brokers might try to dynamically construct composite 'services' consisting of a number of individual services were none of these alone can satisfy the query, or else to apply heuristics to select, negotiate with and invoke services on behalf of the requester. Cf. (Paolucci et al., 2002) for further discussion.

[5] This basic approach can be extended, if more solutions are required, to return instances of classes *which subsume* the query class, or even of those which are merely not necessarily disjoint with the class (although the solutions returned in these cases can no longer be 'guaranteed', in any sense, to satisfy the query).

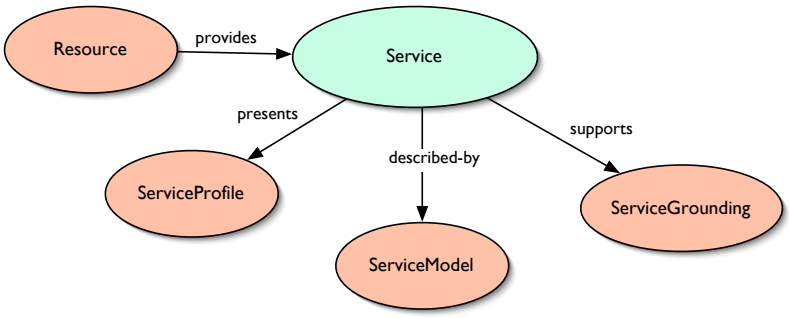

Figure 1: OWL-S Service Ontology

tively uncontroversial to posit a class NL-Resource which is partitioned into two subclasses, NL-StaticResource and NL-ProcessingResource (cf. Cunningham et al., 2000). By 'static resources' we mean things like corpora, probability models, lexicons and grammars; by 'processing resources' (or processors) we mean tools such as taggers and parsers that use or transform static resources in various ways. As mentioned earlier, the main challenge is to find a motivation for imposing a further taxonomy onto NL-ProcessingResource. Our proposal rests on the following ideas:

1. NLP processors have documents as both input and output.
2. Documents have properties which impose preconditions on processors and which also record the effects of processing.
3. A specification of the properties of documents, as input/output parameters, induces a classification of NLP processors.

We make the assumption that NLP tools are in general **additive**, in the sense that they contribute new annotation to an already annotated document and do not remove or overwrite any prior annotation.[6] As a result, at any point in the processing chain, the annotated document is a record of all that has preceded and thereby provides a basis for making subsequent annotation decisions. This general approach is particularly prominent in XML-based approaches to linguistic markup, but is also prevalent elsewhere.

4.1. Document Properties

Figure 2 illustrates the Document class, together with its main properties. We do not wish to be prescriptive about the allowable class of values for each of these properties. Nevertheless, we will briefly describe our current assumptions.

[6] In practice, some removal of low-level annotation might take place, and we could also envisage approaches in which ambiguity is reduced by overwriting previous annotation. Nevertheless, for current purposes the assumption of additivity seems a reasonable simplfication,

Document	
hasMIME-Type	MIME-Type
hasDataFormat	anyURI
hasAnnotation	Annotation
hasSubjectLanguage	ISO-693
hasSubjectDomain	Domain

Figure 2: The Document class

hasMIME-Type: The obvious values to consider are `audio` for processors which allow speech input, and `text/plain` and `text/XML` for text processing tools. However, we also wish to allow cases where the value of hasMIME-Type is underspecified with respect to these second two options. Consequently, we treat Text as a subclass of MIME-Type, partitioned into subclasses TextPlain and TextXML.

hasDataFormat: The value of this property is a URI, more specifically, the URI of a resource which describes the data format of the document. By default, the resource will be an XML DTD or Schema, but any well-defined specification of the document's structure would be acceptable in principle.

hasAnnotation: We treat Annotation as an enumerated class of instances, namely the class {word, sentence, pos-tag, morphology, syntax, semantics, pragmatics}. Although we believe that these annotation types are fairly non-controversial, any broadly-accepted restricted vocabulary of types would be acceptable. The presence of word and sentence reflect the fact that tokenizers will typically segment a text into tokens of one or both these types. Types such as syntax are intended to give a coarse-grained characterization of the dimension along which annotation takes place. However, the specific details of the annotation will depend on the data model and linguistic theory embodied in a given processing step, and we wish to remain agnostic about such details.

hasSubjectLanguage: Following Bird and Simons (2001), we use the term 'subject language' to mean "the language which the content of the resource describes or discusses". Values for this property

are presumed to come from ISO 639 (i.e., two- or three-letter codes).[7]

hasSubjectDomain: We are focussing here on tool-related properties, rather than application-related properties; consequently the domain or subject matter of a document is outside the scope of our discussion. However, within a given application, there may well be domain ontologies which would provide useful detail for this property. Moreover, it is obviously of interest to test whether a statistical tool that has been trained on one domain can be ported to another.

At least some of the document properties that we wish to record fall within the scope of Dublin Core metadata, and indeed we might want augment the properties mentioned above with further elements from the Core, such as `publisher` and `rights`. Bird and Simons (2003) have argued in favour of uniformly building metadata for describing language resources as extensions of the Dublin Core. On the face of it, this is an attractive proposal. However, there is at least a short term obstacle to implementing it within our current framework: as an intellectual resource, an OWL-S ontology also needs to be provided with metadata, and the obvious solution is to encode such information using Dublin Core elements. Thus, we would need to carefully distinguish between metadata concerning the ontology itself, and metadata concerning classes of objects (such as Document) within the ontology. We therefore postpone consideration of this issue to the future.

4.2. Processing Resources

In Figure 3, we sketch a portion of the Profile Hierarchy in order to illustrate the classification of processing resources. The class NL-ProcessingResource is shown with two properties, hasInput and hasOutput: both take values from the class Document. Now, we can create subclasses of Document by restricting the latter's properties. For example, consider the class Document $\sqcap \exists$ hasMIME-Type . Text. This is interpreted as the intersection of the set of things in the extension of Document with the set of things whose hasMIME-Type property takes some value from the class Text.

To create a subclass of NL-ProcessingResource, we restrict the class of the inputs, outputs, or both. For example, if the property hasInput is restricted so that its value space is not the whole class Document, but rather just those documents whose MIME type is Text, then we thereby create a new subclass of NL-ProcessingResource; i.e., those processors whose input has to be text rather than audio. We call this the class NL-Analyzer (implicitly in contrast to speech recognizers, whose input would be audio). Note that since the domain of the property hasMIME-Type is in any case restricted to the class Document, we can simplify hasInput . (Document $\sqcap \exists$ hasMIME-Type . Text) to hasInput . ($\exists$ hasMIME-Type . Text), as shown in the property specification for NL-Analyzer in Figure 3.

[7] Cf. `http://www.loc.gov/standards/iso639-2/`.

Every subclass of NL-Analyzer will of course inherit these restrictions, and will in turn impose further restrictions of their own.[8] Thus, we might insist that every tokenizer identifies and annotates word tokens. That is, NL-Tokenizer's output will be a Document with the additional restriction that the set of annotation types marked in the document contains word. Similarly, NL-Tagger will require that its input document has been marked for the annotation type word (i.e., has been tokenized), and will output a document which has additionally been marked for the annotation type pos-tag.

Recall that as a value of hasMIME-Type, Text is underspecified: it can be specialised as either TextPlain or TextXML. Consequently, a tagger which was able to deal equally with both kinds of input could advertise itself as having the more general value for hasMIME-Type, namely Text. This would allow us to compose the tagger with a tokenizer whose output had the property hasMIME-Type . TextXML—that is, composition is allowed if the input of the tagger subsumes the output of the tokenizer. However, the reverse is not true. Suppose the tagger only accepts input with hasMIME-Type . TextXML. Then it cannot straightforwardly be composed with a tokenizer whose output is more general, namely hasMIME-Type . Text.

Although we have concentrated on Document as the input parameter for processors, we need to allow additional inputs. For example, we allow the NL-Tagger class to have the input parameter usesTagset, where possible instances would include the Penn Treebank Tagset, the CLAWS2 Tagset, and so on. Moreover, the subclass of probabalistic taggers would require an additional input parameter, namely the probability model acquired during training.

Within the framework of OWL-S, we would expect a concrete service to be an instance of a class defined in the Profile Hierarchy. Thus, a particular tagger, say `TnT`, would advertise itself by declaring that it was an instance of NL-Tagger, and further specifying values for properties that were mandatory for this class.

4.3. Data Format Requirements

In our earlier discussion, we said that the value of hasDataFormat would be a file URI. An alternative would be to allow processors to specify abstract data types as inputs and outputs (Sycara et al., 2002; Zaremski and Wing, 1997). For example, we might say that a tagger takes as input a sequence of sentences, each composed of a sequence of word tokens, and outputs a sequence of sentences, each composed of a sequence of word-tag pairs. However this doesn't fit in well with the limitations of ontology languages such as Description Logic. For the purposes of matchmaking, a pointer to a format definition file outside the profile hierarchy seems sufficient and more tractable.

[8] Note that Description Logic, and thus OWL-S, only supports strict inheritance—defaults are not accommodated.

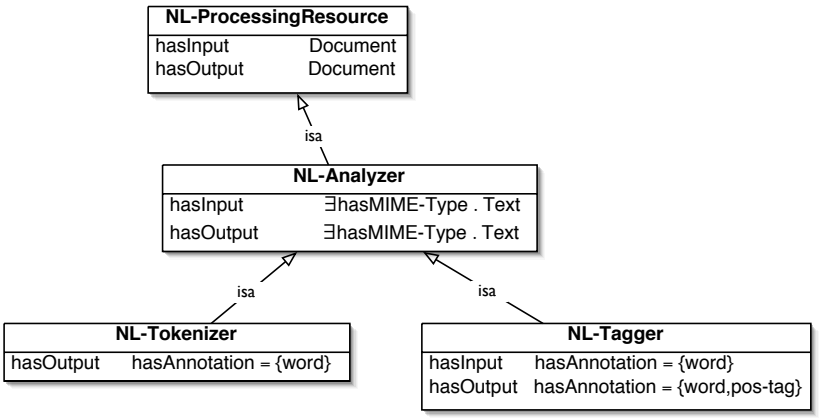

Figure 3: The ProcessingResource class

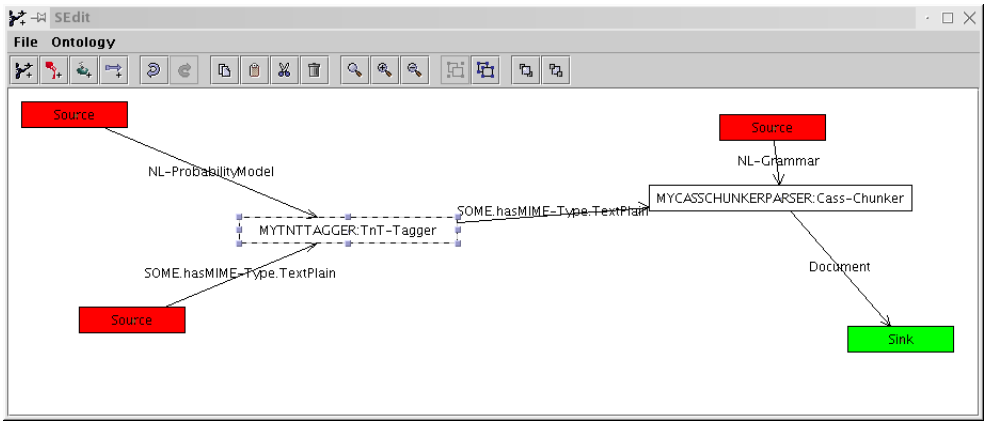

Figure 4: An NLP Web Service Client tool

5. Towards Implementation

To experiment with some of the ideas proposed in this paper, we have developed a prototype environment for the discovery, coordination and (eventually) invocation of NLP Web Services. The NLP Profile Hierarchy described in section 4. has been implemented as an OWL ontology, using the Protégé editor and OWL plugin.[9] Unfortunately, the versions of the OWL-S ontologies available at the time of writing fail to validate in Protégé, and we therefore based our approach on the modified versions made available by Péter Mika at `http://www.cs.vu.nl/~pmika/owl-s/`. The version of the NLP Profile Hierarchy described here can be found at `http://gridnlp.org/ontologies/2004/`.

In order to be able to reason about NLP services, we have used a broker service, built on top of the RACER (Haarslev and Moller, 2001) Description Logic engine. This broker maintains a description, based on the NLP ontology, of the available language processing resources in the environment; when it receives service advertisements, described using OWL-S and this domain ontology, it classifies these and stores them as instances of the appropriate class in the hierarchy. On receiving an OWL-S query, it composes a class description from this and then returns, as potential solutions, (the URLs of) any service instances of classes equivalent to or subsumed by this description.

This broker is itself a web service, accessed through a WSDL end-point. On the client side, we have developed a prototype composition tool for composing sequences of services and querying the broker. The user is able to specify either the type of processing resource that is needed, or the constraints on the data inputs and outputs to some abstract service (or a combination of both) and the tool constructs the appropriate OWL-S, sends this to the broker (via WSDL and SOAP messaging which is hidden from the user) and then presents the alternative services — if any — to the user. Once a user selects one of these, the tool fetches the URL of the service to extract more detailed information about the service, and the user's composition view is updated accordingly.

Figure 4 shows a screen-shot of the tool being used to define a workflow; data, represented by directed edges in this graph (with labels describing the class of the data) flows from 'Sources' to 'Sinks' via one or more services, represented as nodes, labelled with the service name and class. Hence, the screen-shot shows a two-service workflow, producing a DOCUMENT output. To illustrate the use of the tool, and its interaction with the broker, we will now step through the process by which this simple workflow was

[9] See `http://protege.stanford.edu/` for details.

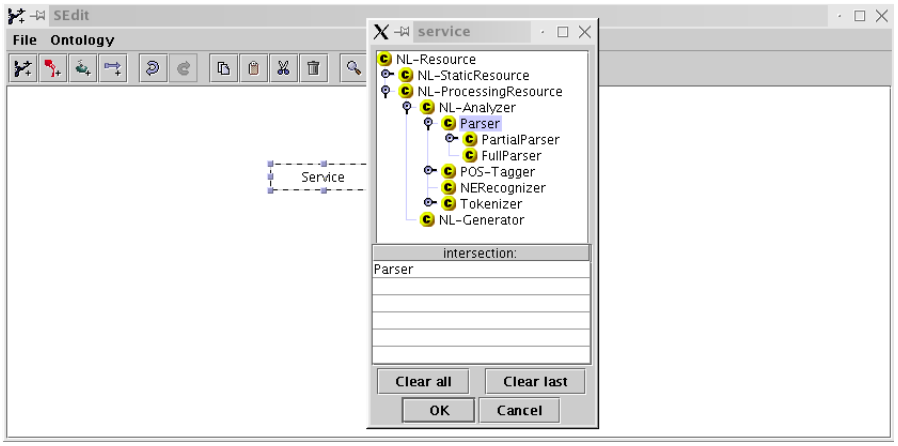

Figure 5: Specifying the class of a desired service

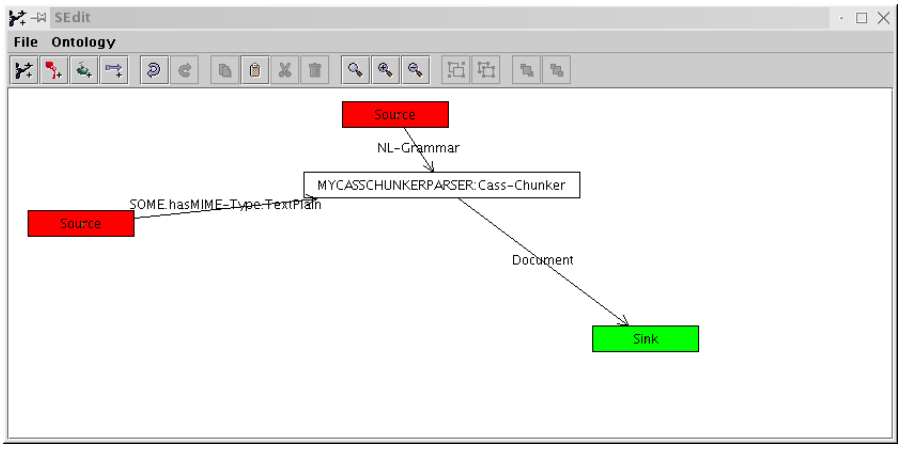

Figure 6: Elaborating the workflow through interaction with the broker

constructed. (To keep this example reasonably clear, the services described are described in rather less detail than might be expected in reality.)

The user — an NLP researcher — here begins with the desire to produce a parsed document. Accordingly, she begins by defining an (anonymous) service of class Parser. The tool has access to the NLP ontology, and a pop-up window allows the class of the desired service to be specified (Figure 5).

Now, the user, via a drop-down menu, places a call to the broker (the address of which is hard-wired into this tool) for details of available services that meet this specification. This has the effect of creating an OWL-S document, the **Profile** of which is an instance of class Parser. Since this is a query, the broker uses this information to find and return the URIs of (the OWL-S descriptions of) all advertised instances of this class and of any of its child classes. In this case, there are two such instances, called **MyLTChunkerParser** and **MyCassChunkerParser**. These are presented to the user as alternatives; she arbitrarily chooses the latter (which is of class Cass-Chunker, an (indirect) subclass of Parser), and its OWL-S description, which specifies the required inputs (namely an NL-Grammar and some (Document) thing which hasMIME-Type of class TextPlain) and the output (a Document), allowing these to be automatically added to the workflow (Figure 6).

Knowing that she needs to first tag the latter Document input, she now replaces its source node with an anonymous service of class POS-Tagger (Figure 7). The broker can now be queried for services of this class which produce an output which hasMIME-Type of class TextPlain. Among the matching services returned by the broker is **MyTNT-Tagger**, which is selected and added to give the workflow shown in Figure 4.

If satisfied with this workflow, the next steps would involve checking and elaborating the workflow further using the OWL-S **Model** of each individual service, and then invoking the workflow using the **Grounding** of each. However, the description of these elements of services will require further conceptualization of the domain, and as a result these steps are not yet implemented. The development of a similar tool to allow human service providers to construct and *advertise* the OWL-S descriptions of their services is also envisaged.

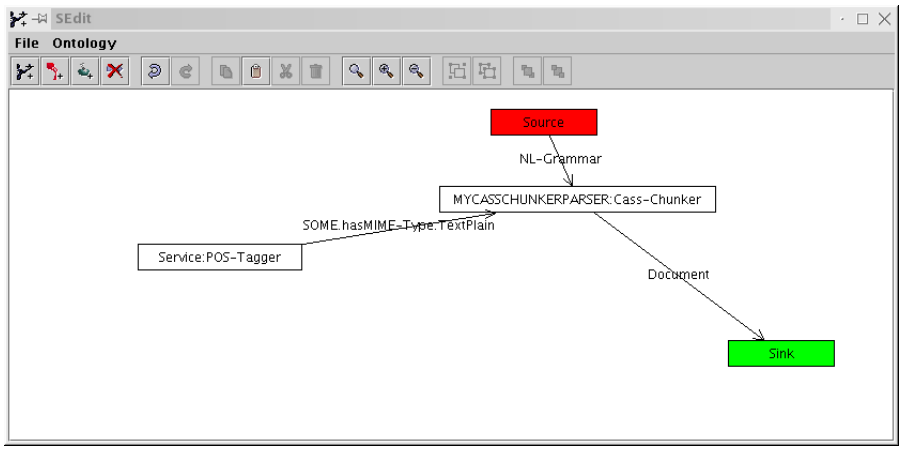

Figure 7: Extending the workflow

6. Conclusion and Future Work

We have argued that a service-oriented view of NLP components offers a number of advantages. Most notably, it allows us to construct an ontology of component descriptions in the well-developed formalism of Description Logic. This in turn supports service discovery and service composition. We have only considered serial composition here, but there is no reason in principle not to allow more complex forms of interaction between components.

One of the most interesting recent frameworks for constructing workflows of NLP components is that proposed by Krieger (2003). His approach deserves more detailed consideration that we have space for here. However, an important difference between our approach and Krieger's is that we do not require components to interact within a specific programming environment such as Java. By wrapping components as services, we can abstract away from issues of platform and implementation, and concentrate instead on the semantics of interoperability. In future work, we will spell out in detail how NLP services described at the OWL-S Profile level can be grounded in concrete resources.

References

Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages for the semantic web. In Dieter Hutter and Werner Stephan, editors, *Festschrift in honor of Jörg Siekmann*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2003.

Tom Bellwood, Luc Clément, and Klaus von Riegen. UDDI technical white paper. `http://uddi.org/pubs/uddi-v3.00-published-20020719.htm`, 2002.

T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

Steven Bird and Gary Simons. The OLAC metadata set and controlled vocabularies. In *Proceedings of the ACL/EACL Workshop on Sharing Tools and Resources for Research and Education*, Toulouse, 2001. Association for Computational Linguistics.

Steven Bird and Gary Simons. Extending Dublin Core Metadata to support the description and discovery of language resources. *Computing and the Humanities*, 37: 375–388, 2003.

Thorsten Brants. TnT – a statistical part-of-speech tagger. In *Proceedings of the 6th Applied NLP Conference, ANLP-2000 of the 6th Applied NLP Conference, ANLP-2000*, 2000. URL `\url{http://acl.ldc.upenn.edu/A/A00/A00-1031.pdf}`.

E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language. `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`, 2001.

Hamish Cunningham, Kalina Bontcheva, Valentin Tablan, and Yorick Wilks. Software Infrastructure for Language Resources: a Taxonomy of Previous Work and a Requirements Analysis. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC-2)*, Athens, 2000.

James Curran and Stephen Clark. Language independent NER using a maximum entropy tagger. In Walter Daelemans and Miles Osborne, editors, *Seventh Conference on Natural Language Learning (CoNLL-03)*, pages 164–167, Edmonton, Alberta, Canada, 2003. Association for Computational Linguistics. In association with HLT-NAACL 2003.

David de Roure and James A. Hendler. E-science: The Grid and the Semantic Web. *IEEE Intelligent Systems*, 19(1): 65–70, January/February 2004.

Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, 2002.

Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, and H. F. Nielsen. Simple object access protocol (SOAP). `http://www.w3.org/TR/soap12-part1/`, 2003.

V. Haarslev and R. Moller. RACER system description. In *Proceedings of the First International Joint Conference on Automated Reasoning*, pages 701–706. Springer-Verlag, London UK, 2001.

Tony Hey and Anne E. Trefethen. The UK e-Science core programmed and the Grid. *Future Generation Computer Systems*, 18(8):1017–1031, 2002. ISSN 0167-739X.

Hans-Ulrich Krieger. SDL—A description language for building NLP systems. In Hamish Cunningham and Jon Patrick, editors, *HLT-NAACL 2003 Workshop: Software Engineering and Architecture of Language Technology Systems (SEALTS)*, pages 83–90, Edmonton, Alberta, Canada, May 2003. Association for Computational Linguistics.

OWL-S. OWL-S: Semantic markup for web services. `http://www.daml.org/services/owl-s/1.0/`, 2003.

Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic matching of web services capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC2002)*, pages 333–347, 2002.

Katia Sycara, Seth Widoff, Matthias Klusch, and Jianguo Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5:173–203, 2002.

Amy Moormann Zaremski and Jeannett M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4): 333–369, 1997.