Developing a Service-Oriented Architecture to Harvest Information for the Semantic Web

Barry Norton, Sam Chapman and Fabio Ciravegna

Department of Computer Science, University of Sheffield
Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK
{B.Norton, S.Chapman, F.Ciravegna}@dcs.shef.ac.uk

Abstract. Armadillo is a tool that provides automatic annotation for the Semantic Web using unannotated resources like the existing Web for *information harvesting*, that is: combining a *crawling* mechanism with an extensible architecture for *ontology population*. The latter is achieved via largely unsupervised machine learning, boot-strapped from oracles, such as web-site wrappers, and backed up by an 'evidential reasoning', allowing evidence to be gained from the redundancy in the Web and allowing the inaccuracies in information, also characteristic of today's Web, to be circumvented. In this paper we sketch how the Armadillo architecture has been reinterpreted as workflow templates that compose semantic web services and show how the porting of Armadillo to new domains, and the application of new tools, has thus been simplified.

1 Introduction

The Semantic Web needs semantically-based document annotation to both enable better document retrieval and empower semantically-aware software agents. Most of the current technology is based on human centred annotation, very often completely manual which can be incorrect and incomplete, deliberately or through lack of skill, or can become obsolete. A major advantage of Armadillo [2] is its ability to annotate large repositories in a largely unsupervised manner and thereby make available to the Semantic Web the huge amount of information, available only in human-oriented form, on the current Web.

Armadillo annotates by extracting information from different sources, boot-strapped by 'oracles' i.e. relatively infallible authorities. 'Evidential reasoning' is used to validate the classifications of, and relations between, instances. This evidence is then integrated and the knowledge entered into a repository.

Our aim in using a semantic web service (SWS)-based architecture is to allow porting to a new domain by providing workflow templates, *i.e.* where some subtasks are left as parameters, expressed in BPEL [5], where SWS's achieving these subtasks are described in OWL-S [4]. We claim that this allows the process to be understood abstractly, and represented graphically rather than in code, and that those points where services must be located and 'plugged in' are more easily understood in terms of the OWL [8] concepts they relate. Furthermore direct reuse of domain-specific and externally-authored functionality is facilitated.

2 Architecture

The act of porting Armadillo to a new ontology population task begins by providing a domain-specific ontology. The given ontology includes structural relationships which are transformed into a plan for population. The plan details an order in which the concepts and relations will be explored by attaching a direction to each relation. For each relation to be followed, from a concept A to a concept B, the workflow shown in Figure 1 organises the following subtasks: Crawling, Instance Recognition, Evidential Reasoning and finally Combination and Storage. Crawling will systemically retrieve documents associated with an instance of concept A; Instance Recognition will find candidate instances of B with an implicit relation to that instance; Evidential Reasoning will find support for both the classification and the relation discovered, which will be combined and, if sufficient, cause both to be stored.

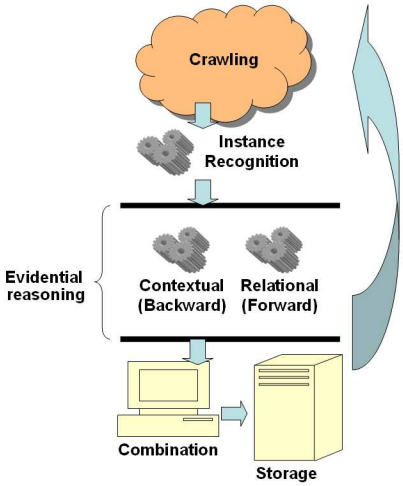

Fig. 1. Architecture

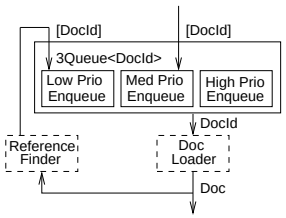

Fig. 2. *Crawling* Task

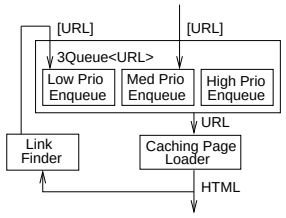

Fig. 3. Example *Crawling* Instantiation

In the remainder of the paper we detail the template workflows which achieve each of these subtasks in turn, composing generic services with others that a developer must locate to accomplish clear strategies. For these the choice, aided by *semantic discovery* [7], is between:

- *generic* services: wholly independent of domain and context;
- *context-dependent* services: where reuse depends on the context but not the domain (*e.g.* the type of documents or repository used);
- *domain-tailored* services: parameterised to be tailored to a given domain;
- *domain-trained* services: encapsulate machine-learning which can adapt semi-automatically to a given domain;
- *domain-specific* services: encapsulate techniques which are hard-coded for a given domain (but might still be re-used across applications in that domain).

In the following we shall use a familiar example of the academic domain [2], wherein we view instances of the 'University' concept as being provided by an oracle and populate the 'Academic' concept, via the 'employedBy' relationship. We use square brackets to represent lists over the contained type, and so $[DocId]$ is the type of a list of elements of type $DocId$. Angle brackets mean instantiation of some generic (parameterised) service, so that $3Queue$ represents some generic queue service that stores instances of some type notated $DocId$, at three different priority levels, and supplies them to the consequent workflow one at a time.

2.1 Crawling

The general form of the 'crawling' task is shown in Figure 2. There are two levels at which this abstract workflow is parameterised: the $DocId$ and Doc labels are actually type variables that must be instantiated at concrete types; the $DocLoader$ and $ReferenceFinder$ tasks are service variables to be instantiated.

Figure 3 shows how we could achieve a concrete instantiation of this task. First we choose instances for the type variables consistent with Web-oriented technology, i.e., in this example, $Doc = HTML$ and $DocId = URL$. We then choose *context-dependent* services that meet the resulting signatures, i.e. producing a page from a URL, and a list of URLs from a page, respectively.

2.2 Instance Recognition

The 'instance recognition' task can be realised by both *domain-tailored*, and *domain-trained* services to find candidate instances of a given type B with an implicit relation to the instance of A being investigated. This can be seen in Figure 4, where the $B{-}Recogniser$ outline implies that multiple parallel implementations may be used. For instance in the example in Figure 5 we see that both *domain-tailored* regular expression matching and *domain-trained* Amilcare [3] will be used side-by-side, as could other Information Extraction tools.

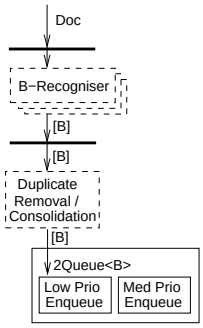

Fig. 4. *Instance Recognition* Task

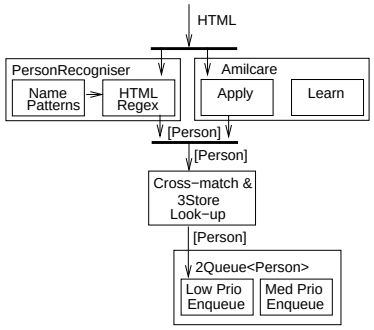

Fig. 5. Example *Recognition* Instantiation

The subsequent consolidation stage is typically domain-specific Information Integration but reuse can be made; for instance from the 'similarity metrics' library, *SimMetrics*, which we have recently released [1]. The consolidated list of candidate instances is then queued to be validated by 'evidential reasoning'.

2.3 Evidential Reasoning

For each potential instance of the concept B from the queue this task will attempt to find evidence to confirm its classification and relation to the original instance. This may consist of several parallel services, each of which implements a strategy that falls into one of two classes, as follow.

Contextual Reasoning considers each potential B instance in the context of the A instance via which it was discovered. As seen in Figure 6, two services will be used to find occurrences of the B instance in general and co-located with the A instance respectively. A third service then produces a list of potential triples relating these instances, with evidence supporting the relation. Figure 7 shows a simple instance of this strategy where we 'promote' the candidate instance to being an academic employed by the university based on co-located references on the web, obtained by a Google wrapper which is domain-independent.

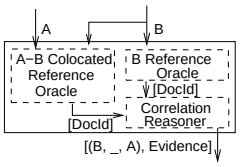

Fig. 6. *Contextual Reasoning* Task

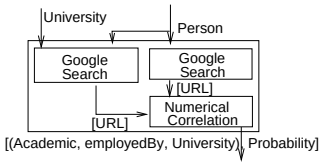

Fig. 7. Example *Contextual* Instantiation

Relational Reasoning provides evidence for the candidate B instance being correctly classified as such, based on other relations an oracle may find. In this subtask, as shown in Figure 9, we may apply *domain-tailored* or *domain-specific* technologies such as gazetteers and site wrappers, as well as *domain-trained* relation extraction, as we are developing in the tool *T-Rex*.

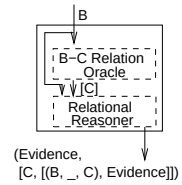

Fig. 8. *Relational Reasoning* Task

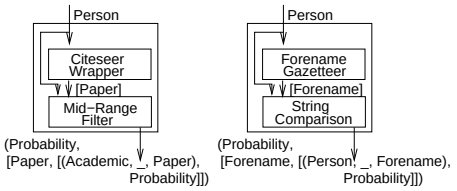

Fig. 9. Example *Relational* instantiations

2.4 Combination and Storage

For the sake of brevity we do not consider, in this paper, the 'combination' and 'storage' tasks except to say that we have instantiated the *Evidence* type variable as a probability and will use a statistical combination to provide a higher-confidence result; other solutions are also accommodated.

3 Conclusions

We have illustrated how it is possible to make use of SWS's in harvesting the Web, and other corpora, to provide annotations for the Semantic Web. The architecture presented is based on workflow and follows an IE-oriented strategy. Initial approximations to both classification and implicit relation extraction are followed by evidential reasoning based on both context and further relations. In this way a wide variety of semantic web services may be accommodated and porting is eased since, in many cases, users can avoid coding altogether, merely using the workflow templates to guide semantic discovery and composition.

This architecture also provides many other benefits associated with service-oriented architectures, such as speed-up from concurrency and distribution, an automatic means to reuse of any code that does have to be written specifically for a new domain, and the ability to provide services remotely to users with little infrastructure. A more complete picture of the workflow template described here, and the means by which it is currently implemented, is provided in the related position paper [6].

References

1. Sam Chapman. SimMetrics. http://sourceforge.net/projects/simmetrics/.
2. Fabio Ciravegna, Sam Chapman, Alexiei Dingli, and Yorick Wilks. Learning to harvest information for the semantic web. In *Proceedings of the First European Semantic Web Symposium*, May 2004.
3. Fabio Ciravegna and Yorick Wilks. *Annotation for the Semantic Web*. Series Frontiers in Artificial Intelligence and Applications. IOS Press, 2003.
4. Anupriya Ankolekar *et al.* DAML-S: Web service description for the semantic web. In *Proc. 1st International Semantic Web Conference (ISWC)*, 2002.
5. IBM *et al.* Business process execution language for web services version 1.1. http://www-128.ibm.com/developerworks/library/ws-bpel, 2003.
6. Barry Norton. Proposed functional-style extensions for semantic web service composition. In *Proc. 1st AKT Workshop on Semantic Web Services*, 2004.
7. Naveen Srinivasan, Massimo Paolucci, and Katia Sycara. Adding OWL-S to UDDI: implementation and throughput. In *Proc. 1st Intl. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, pages 6–9, 2004.
8. W3C. OWL web ontology language overview. http://www.w3c.org/TR/owl-features, 2004.

Acknowledgements

This work was carried out within the AKT project (http://www.aktors.org), sponsored by the UK Engineering and Physical Sciences Research Council (grant GR/N15764/01), and the Dot.Kom project, sponsored by the EU IST asp part of Framework V (grant IST-2001-34038).