Web Services Architecture for Language Resources

Angelo Dalli*, Valentin Tablan*, Kalina Bontcheva*, Yorick Wilks*, Dan Broeder, Hennie Brugman**, Peter Wittenburg****

* NLP Research Group, Department of Computer Science
University of Sheffield
{a.dalli, v.tablan, k.bontcheva, y.wilks}@dcs.shef.ac.uk

** Max Planck Institute for Psycholinguistics, Nijmegen
{dbroeder, hbrugman, pwittenburg}@mpi.nl

Abstract

A web services based architecture for Language Resources utilizing existing technology such as XML, SOAP, WSDL and UDDI is presented. The web services architecture creates a pervasive information infrastructure that enables straightforward access to two kinds of Language Resources: traditional information sources and language processing resources. Details about two practical implementations of this web services architecture are given.

Web Services

The concept of web services as being lightweight components that offer an elegant means of integrating different information repositories and services across the Internet has always been a main objective in developing a standard, interoperable system of web services. Industrial and academic support for web services is increasingly gaining strength and the future looks promising for their widespread adoption (Narsu and Murphy, 2002; Conner, 2001; Gates, 2003). The idea of using web services for Computational Linguistics is also gaining acceptance with the increasing availability of various useful services permitting researchers unprecented access to huge amounts of information and advanced search services like Google (Google, 2002). Linguistic resources are prime candidates for web services applications to enable increased collaboration between research groups and avoid reduplication of resources and effort. Fortunately, current web services technology can be used to provide effective solutions to common problems faced by researchers (Dalli, 2001; Dalli, 2002).

We propose a web services architecture for Language Resources that uses a combination of Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) and Universal Discovery Description Integration (UDDI) to achieve maximum benefit from these technologies in a Computational Linguistics context (Box et al., 2000; Christensen, et al., 2001; UDDI, 2001). The web services architecture creates a pervasive information infrastructure that enables straightforward access to two kinds of Language Resources: traditional resources such as lexicons, corpora, semantic networks, etc. and language processing resources. The use of standard technology ensures that there is wide support for developers working with minimal knowledge of web services, and also guarantees compatibility with legacy applications, while keeping compatibility with major development frameworks such as Sun's Java, IBM's WebSphere, and Microsoft's .NET.

Existing Technologies

Most Language Resources that are currently available for research and development can be currently classified as a heterogeneous collection of different proprietary formats and databases with minimal means, if any, of interoperability with other Language Resources making it hard to extend their usefulness beyond the life of their originating projects (Cunningham, 1999). This is an even more serious issue for smaller projects and Language Resources for minority languages, since fewer people will be willing to utilize non-major Language Resources if there is no commonly accessible metadata description that enables established tools to be used in an interoperable manner.

The web services architecture achieves the goals of a pervasive information infrastructure by using WSDL as its Language Resource metadata description language, UDDI as its main publication and discovery mechanism, and SOAP as the means to retrieve information and execute remote processes.

One main feature of these technologies is their reliance on the availability of XML marked up data. Fortunately, a significant amount of Language Resources are already in a compatible format such as linguistic data marked up in the Resource Description Framework (RDF) (Lassila and Swick, 1999; Klyne et al., 2003), Open Lexicon Interchange Format (OLIF) (McCormick, 2002), XCES-EAGLES-ISLE format (Zampolli, 2000; EAGLES, 2000; Bertagna et al., 2000) and Encoded Archival Description (EAD) (NDMSO, 2002). A conversion layer using the Extensible Stylesheet Language (XSL) or some other appropriate technology can easily convert this kind of information into the required XML format.

The web services architecture will need a common taxonomy, called the Interoperable Extensible Language Resource (IELR) standard, that caters for the most common subset of linguistic information used to markup and classify language resources, together with a similar component for language processing resources. Due to the lack of adequate taxonomies for language processing resources IELR will adapt work done in related projects, namely the General Architecture for Text Engineering (GATE) and the Open Archives Initiative (OAI) (OAI, 2001; Wilks et al., 1998).

Legacy Applications and Interoperability

Legacy applications will need to have a custom made data conversion layer to ensure that the data can be converted into some IELR compatible format. The amount of effort required for this conversion layer depends on the degree of structure present in the original format. An extensions interface in the web services architecture allows legacy applications, non-standard extensions such as uncommon language phenomena and entirely new classes of language processing techniques to be accommodated transparently. The conversion layer implementation for legacy Language Resources will thus need to provide necessary transformations that convert proprietary formats to the standard core format expected by the SOAP server, while doing this transformation in reverse to facilitate updates of the Language Resource by other linguistic applications and processes.

Interoperability and inter-process communication are achieved through SOAP. SOAP is used to encapsulate all IELR resources, acting as a means of accessing relevant information. SOAP provides XML-based interactions between different Language Resources and related applications over the HTTP protocol. Additionally, SOAP also permits applications to run appropriate processes on remote computers, using remotely stored data. Remote process execution on Linguistic Resources is an area that is still largely undeveloped in the Computational Linguistics community. Initiatives such as the amalgamation of grid based computing and web services will hopefully bring substantial benefits, enabling more sophisticated large-scale text processing to be performed.

The web services architecture defines a set of criteria for applications to expose their underlying processing algorithms – basically services have to support local and remote data access, need to support pausing and resuming of their processes, and need to be capable of splitting up large data processing requests into small manageable steps. This flexible approach ensures that applications that conform to the web services architecture specification can implement their own methods for scheduling and resource use optimization.

Figure 1 shows how the SOAP layer is used in conjunction with the conversion layer to provide an effective encapsulation of the Language Resource, enabling a common format for all Language Resources to be expected by applications designed to run on the web services architecture.

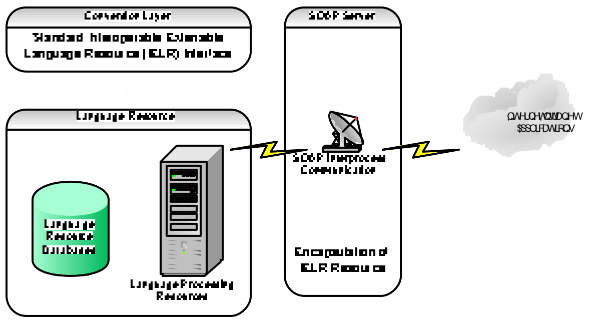
Conversion Layer
Standard Interoperable Extensible Language Resource (IELR) Interface
SOAP Server
Language Resource
Language Resource Database
Language Processing Resources
SOAP Interprocess Communication
Encapsulation of IELR Resource

Figure 1 SOAP used for LR interoperability

Currently, few NLP applications and frameworks support distributed processing and the notion of processing resources. The popular GATE architecture actually has some support for remote process execution and algorithm abstraction, but this remains an under-researched area in Computational Linguistics.

Additionally, initiatives such as the Open Archives Initiative (OAI) and GATE already solve many of the problems that arise in ensuring interoperability and metadata descriptions of services and content, making them both suitable for the implementation of diverse Language Resources. The main drawback to these two solutions is their reliance on proprietary data formats and protocols, making it difficult for other third-party applications to readily interoperate with these architectures.

Performance considerations were taken into account in the design of the architecture with communication overhead and network congestion identified as being the two most serious bottlenecks. Although XML is the natural choice for storing and representing linguistic data due to its simplicity and compatibility with a variety of existing systems, its main drawback is that pure XML databases are usually limited in their performance due to the significant amount of processing needed to encode and decode huge datasets in what is essentially a pure text format. A better performance solution in this case was found to be to utilise a two-pronged strategy where more efficient data transfer protocols are used in preference to HTTP for content delivery, and using traditional RDBMS technology instead of pure XML datasets to speed up processing. Relation database records can be used to store linguistic data efficiently with a simple transformation method converting the relational data to XML format.

LR Metadata Descriptions and LR Discovery

WSDL provides a standard means of creating accessible XML based metadata descriptions of the Language Resource being abstractly represented by the SOAP server. WSDL is used to add an abstract layer describing the services and features provided by the Language Resource in a standard manner, significantly reducing the development time for new applications and related information extraction and analysis programs. Additionally, client applications using WSDL are shielded from the server implementation, greatly simplifying maintenance and upgrade of existing facilities.

Figure 2 shows how WSDL acts as a metadata description layer for the SOAP-encapsulated Language Resource. WSDL provides a comprehensive means of describing the mechanisms that should be used to access and process content pertaining to a specific Language Resource. A set of abstract operations – that can either return unprocessed or processed information – are bound to some network protocol and finally assigned to some physical address to create a WSDL port. A series of WSDL ports are then packaged together to form a web service.

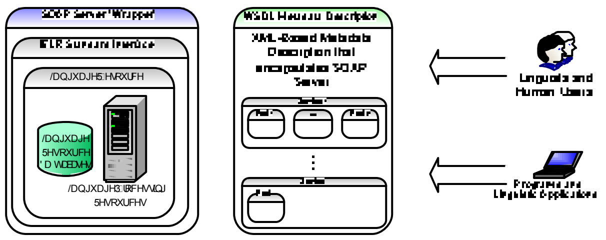
SOAP Server 'Wrapper'
IELR Standard Interface
WSDL Metadata Description
XML-Based Metadata Description that encapsulates SOAP Server
Linguists and Human Users
Programs and Linguistic Applications

Figure 2 WSDL used for LR extensibility

UDDI provides the third key component in the web services architecture. UDDI provides a global registry of web services that facilitates the development of an International Language Resource Directory that aids in the dissemination of metadata descriptions across different research projects. UDDI makes it possible for research projects around the world to find relevant Language Resources easily according to a particular language or linguistic phenomenon, and also according to the type of processing resource needed.

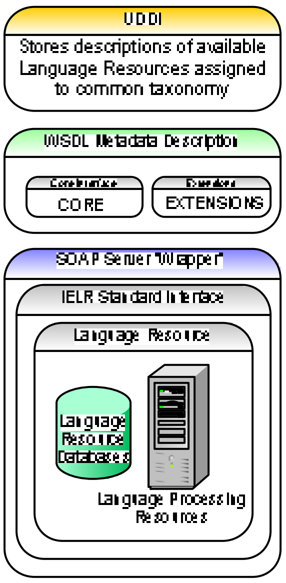
UDDI
Stores descriptions of available Language Resources assigned to common taxonomy
WSDL Metadata Description
CORE
EXTENSIONS
SOAP Server "Wrapper"
IELR Standard Interface
Language Resource
Language Resource Databases
Language Processing Resources

Figure 3 UDDI used for universality and automated LR discovery and integration

A common taxonomy for Language Resources, especially for Language Processing Resources, is still not readily available. Prototype taxonomies were developed for our practical development experiments, but significant work is envisaged to get the computational linguistics community to agree on a standard UDDI taxonomy that enables Language Resources created by various projects to be classified and matched up accordingly using the automated search functions already provided by the UDDI servers.

Prototype Applications

Prototype applications of the web services architecture have been made for two applications at the Max Planck Institute for Psycholinguistics and the University of Sheffield, in a data-oriented and processing-oriented context respectively. These prototype applications enabled the theoretical framework of the web services architecture to be implemented in practice, gaining additional insights into the process, while verifying the ease with which web services can be added to existing applications.

Controlled Vocabulary Service

The controlled vocabulary service was implemented at the Max Planck Institute for Psycholinguistics using a team of two developers over the course of two days, with an additional day for testing and integration. The MPI application involved adding a web service on top of a controlled vocabulary application, related to the IMDI project, which enables researchers to find and exchange relevant controlled vocabularies using web services (Wittenburg et al., 2000). The main problem was in enabling new users of the controlled vocabulary application to automatically publish and share their controlled vocabularies with other users. Since the data was already stored in a standard XML-based format, the effort focused on utilizing the publication and search features provided by the web services architecture to create an effective solution to this problem.

The service was implemented using the Java Web Services Development Toolkit (JWSDP). JWSDP provides the basic packages needed to add web services to any Java application. Four additional sub-packages were needed, namely the Java XML processing package (JAXP), XML-based RPC (JAX-RPC), Java SOAP-based messaging methods (JAXM) and the Java UDDI registry access methods (JAXR). The main process involved the creation of a WSDL stub on both the server side and the client side. The service definition was initially defined and then implemented to create the WSDL stubs using the automated mapping tool provided in JWSDP. The web application components, together with helper classes and static resources needed to support the implementation were created to obtain the final Web Application Archive (WAR) file needed to deploy the web service.

The main difficulty encountered in the MPI implementation was the fact that minimal support existed for Java web services at the time when this prototype was created in 2002. The situation has somewhat improved with the availability of more sophisticated tools that reduce the amount of steps and integration needed, as seen in our second prototype development. Some problems were also encountered in the integration of our prototype taxonomy with UDDI, although this was expected since it was the first time UDDI was used for this purpose.

GATE ANNIE Service

At the University of Sheffield, the web services architecture was implemented on top of GATE's ANNIE component, which aims to eliminate the need for users to keep re-implementing frequently needed algorithms and provide a good starting point for new applications (Cunningham et al., 2002; Maynard, 2002). The service was implemented using the Apache Axis Toolkit, which provides a web services extension to the popular Apache server, enabling existing web based applications to utilised web services with minimal redevelopment effort.

In order to lower the application development overheads, GATE provides a number of useful and easily customizable components, grouped together to form the ANNIE (A Nearly-New Information Extraction) component[1]. These components eliminate the need for users to keep re-implementing frequently needed algorithms and provide a good starting point for new applications. The majority of these components use GATE's finite state techniques to implement various tasks from tokenisation to semantic tagging and coreference, with an emphasis on efficiency, robustness, and low-

[1] A demonstration of how these components can be used to highlight information in Web pages is available at http://gate.ac.uk/annie/index.jsp

overhead portability, rather than full parsing and deep semantic analysis. GATE also provides an extendable set of document format handlers (e.g., XML, HTML, RTF, email), which translate the document content and the formatting information into GATE's shared data model, in a similar way to the conversion layer in our web services architecture.

GATE's graphical development environment also enables the user to create and store GATE applications, so that GATE can load and configure all modules automatically at subsequent executions. Users choose which processing resources go into their application (e.g. tokeniser, POS tagger), in what order they will be executed, and on which data (e.g. document or corpus). For example, ANNIE is a stored GATE application which when selected for loading, automatically loads and configures all its components. The GATE Web services API uses GATE applications to allow users to turn their applications automatically into web services, by providing their stored application to the server.

The effort involved in using the Axis toolkit to create the GATE web services API encapsulating ANNIE was considerably simpler than with using JWSDP on its own. A GATE web service interface was defined with various methods corresponding to ANNIE's methods. Stub implementations were provided for all ANNIE methods, and a WSDL file was generated automatically together with an Apache Axis configuration file. After the server application was built, a corresponding client application was built into GATE to consume web service calls provided by other GATE implementations, effectively turning GATE into both client and server simultaneously.

Conclusion

The experience gained from applying theory in practice shows that although web services still need to improve in certain areas, they can provide useful results in a short time, with two days of actual development time being needed in both prototype experiments. The two prototype experiments also provided useful insights as to how both traditional data-oriented (MPI) and processing-oriented (GATE) Language Resources can be encapsulated seamlessly along with data resources to form a truly accessible Language Resource.

The low learning curve and minimal costs involved in integrating these technologies into existing projects makes the proposed system highly attractive for small and medium sized projects that have limited available resources.

References

Bertagna, F. Calzolari, N. Lenci, A. Zampolli, A. (2001). The Multilingual ISLE Lexicon Entry (MILE). ISLE Computational Lexicons Working Group Report, Italy.

Box, D. et al. (2000). Simple Object Access Protocol 1.1. W3C Note. http://www.w3.org/TR/SOAP

Christensen, E. et al. (2001). Web Services Description Language 1.1. W3C Note. http://www.w3.org/TR/wsdl

Conner, M. (2001). Web Services: The next horizon for e-business, International Business Machines Corporation, Executive Presentation, Armonk, New York.

Cunningham, H. (1999). A Definition and Short History of Language Engineering. Journal of Natural Language Engineering, Cambridge University Press, 5:1-16.

Cunningham, H. Maynard, D. Bontcheva, K. Tablan, V. (2002). GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In Proceedings 40th Anniversary Meeting of the Association for Computational Linguistics (ACL 2002). Budapest.

Dalli, A. (2001). Interoperable Extensible Linguistic Databases, In Proceedings IRCS Workshop on Linguistic Databases, University of Pennsylvania, Philadelphia.

Dalli, A. (2002). Creation and Evaluation of Extensible Language Resources for Maltese, In Proceedings 3rd International Conference on Language Resources and Evaluation (LREC) 2002, Las Palmas de Gran Canaria, Spain.

Expert Advisory Group on Language Engineering Standards (EAGLES). (2000). Corpus Encoding Standard for XML. Vassar College, New York. Equipe Langue et Dialogue LORIA/CNRS, France.

Gates, B. (2003). Keynote speech at Microsoft Professional Developers Conference 2003 (PDC), 27 October 2003, Los Angeles, California.

Google Inc. (2002). Google Web API, Technical Documentation, Mountain View, California. http://www.google.com/apis

Klyne, G. Carroll, J. McBride, B. (2003). Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/TR/rdf-concepts

Lassila, O. Swick, R. (1999). Resource Description Framework (RDF) Model and Syntax Specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222

Maynard, D. Tablan, V. Cunningham, H. Ursu, C. Saggion, H. Bontcheva, K. Wilks, Y. (2002). Architectural elements of language engineering robustness. Journal of Natural Language Engineering, Special Issue on Robust Methods in Analysis of Natural Language Data, 8(2/3):257–274.

McCormick, S. (2002). Open Lexicon Interchange Format. OLIF Consortium. http://www.olif.net

Narsu, U. Murphy, P. (2002). Web Services adoption outlook improves. Giga Information Group, Report RPA-042002-00011.

Network Development & MARC Standards Office (NDMSO). (2002). Encoded Archival Description, Library of Congress, Washington D.C.

Open Archives Initiative (OAI). (2001). The Open Archives Initiative Protocol for Metadata Harvesting. http://www.openarchives.org

UDDI Consortium. (2001). UDDI Version 2.0 Data Structure Reference. http://www.uddi.org/pubs

Wilks, Y. Gaizauskas, R. Cunningham, H. (1998). GATE: General Architecture for Text Engineering. University of Sheffield, Sheffield.

Wittenburg, P. Broeder, D. Sloman, B. (2000). Documentation of Languages and Archiving of Language Data at the Max Planck Insitute for Psycholinguistics in Nijmegen. Presented at "Ringvorlesung Bedrohte Sprachen" Sprachenwert – Dokumentation, Revitalisierung. Fakult?t fur Linguistik und Literaturwissenschaft, Universit?t Bielefeld

Zampolli, A. (2000). Extensions of PAROLE & SIMPLE resources: National Projects. SIMPLE: From Monolingual to Multilingual Resources Workshop, Athens.