A General Architecture for Connecting NLP Frameworks and Desktop Clients using Web Services

René Witte and Thomas Gitzinger

[1] Department of Computer Science and Software Engineering
Concordia University, Montréal, Canada
[2] Institute for Program Structures and Data Organization (IPD)
University of Karlsruhe, Germany

Abstract. Despite impressive advances in the development of generic NLP frameworks, content-specific text mining algorithms, and NLP services, little progress has been made in enhancing existing end-user clients with text analysis capabilities. To overcome this software engineering gap between desktop environments and text analysis frameworks, we developed an open service-oriented architecture, based on Semantic Web ontologies and W3C Web services, which makes it possible to easily integrate any NLP service into client applications.

1 Introduction

Research and development in natural language processing (NLP), text mining, and language technologies has made impressive progress over the last decade, developing innovative strategies for automatically analyzing and transforming large amounts of natural language data. Machine translation, question-answering, summarization, topic detection, cluster analysis, and information extraction solutions, although far from perfect, are nowadays available for a wide range of languages and in both domain-dependent and -independent forms [1].

Despite these advances, none of the newly developed technologies have materialized in the standard desktop tools commonly used by today's knowledge workers—such as email clients, software development environments (IDEs), or word processors. The vast majority of users still relies on manual retrieval of relevant information through an information retrieval tool (e.g., Google or Yahoo!) and manual processing of the (often millions of) results—forcing the user to interrupt his workflow by leaving his current client and performing all the "natural language processing" himself, before returning to his actual task.

This lack of integration is not caused by missing NLP services, nor can it be attributed to a lack of interest by end users: Rather, a gap exists between the domains of software engineers (developing end-user clients) and language engineers (implementing NLP services) that so far has been keeping these worlds apart. One approach seen in the past is the development of new (graphical) user interfaces for a particular analysis task from scratch. While this allows for highly

specialized clients, it also requires significant development efforts. Furthermore, today's users, who face constant "information overload," particularly need support within clients commonly used to develop and access natural language content—such as email clients or word processors. Reengineering all these existing clients just to deliver NLP services (like question-answering or summarization) is not a feasible approach.

The solution proposed here is an open service-oriented architecture (SOA) based on established standards and software engineering practices, such as W3C Web services with WSDL descriptions, SOAP client/server connections, and OWL ontologies for service metadata, which together facilitate an easy, flexible, and extensible integration of any kind of language service into any desktop client. The user is not concerned with the implementation or integration of these services, from his point of view he only sees context-sensitive *Semantic Assistants* relevant for his task at hand, which aid him in content analysis and development.

2 Related Work

Some previous work exists in building personalized information retrieval agents, e.g., for the Emacs text editor [2]. These approaches are typically focused on a particular kind of application (like emails or word processing), whereas our approach is general enough to define NLP services independently from the end-user application through an open, client/server, standards-based infrastructure.

The most widely found approach for bringing NLP to an end user is the development of a new interface (be it Web-based or a "fat client"). These applications, in turn, embed NLP frameworks for their analysis tasks, which can be achieved through the APIs offered by frameworks such as GATE [3] or UIMA [4]. The BioRAT system [5] targeted at biologists is an example for such a tool, embedding GATE to offer advanced literature retrieval and analysis services. In contrast with these approaches, we provide a service-oriented architecture to broker any kind of language analysis service in a network-transparent way. Our architecture can just as well be employed on a local PC as it can deliver focused analysis tools from a service provider.

Recent work has been done in defining Web services for integrating NLP components. In [6], a service-oriented architecture geared towards the field of terminology acquisition is presented. It wraps NLP components as Web services with clearly specified interface definitions and thus permits language engineers to easily create and alter concatenations of such components, also called processing configurations. This work can be seen as complimentary to our approach, since the granularity of our ontology model does not extend to individual NLP components, but rather complete NLP pipelines (applications build from components).

3 The General Integration Architecture

In this section, we present an overview of our architecture integrating NLP and (desktop) clients, delivering "Semantic Assistants" to a user.

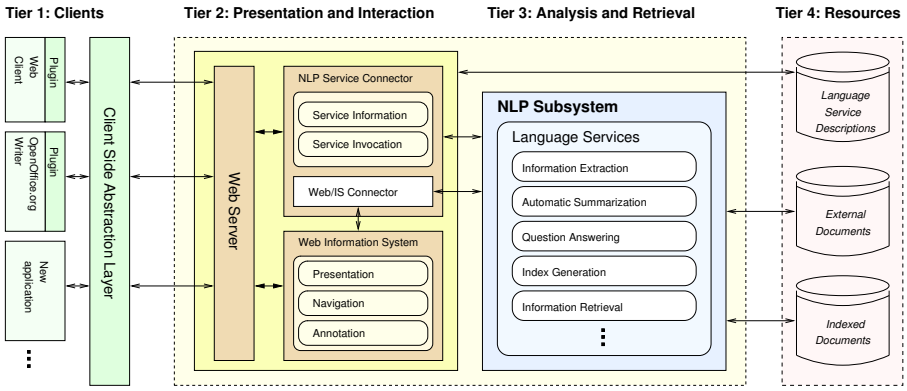

Fig. 1. Service-oriented architecture for integrating NLP and end-user clients

3.1 Architectural Overview

Our architecture is based on a multi-tier information system design (Fig. 1).

Tier 1: Clients. This tier has the main purpose of providing access to the system. Typically, this will be an existing client (like a word processor or email client), extended to connect with our architecture through a plug-in interface. Besides facilitating access to the whole system, clients are also in part responsible for presentation, e.g., of language service results. In addition to the actual client applications, an abstraction layer is part of Tier 1. It shields the clients from the server and provides common functionality for NLP services.

Tier 2: Presentation and Interaction. Tier 2 consists of a standard Web server and a module labeled "NLP Service Connector" in the diagram. One responsibility of this module is interaction, in that it handles the communication flow between the NLP framework and the web server. Moreover, it prepares language service results, by collecting output from the NLP services and transforming them into a format suitable for transmission to the client. Finally, the NLP Service Connector reads the descriptions of the language services, and therefore "knows" the vocabulary used to write these descriptions. In addition, our architecture can also provide services to other (Web-based) information systems, like a Wiki [7].

Tier 3: Analysis and Retrieval. Tiers 1 and 2 are the ones the user has direct contact with. Tier 3 is only directly accessed by the NLP Service Connector. It contains the core functionality that we want, through Tiers 1 and 2, to bring to the end user. Here, the language services reside, and the NLP subsystem in whose environment they run (such as GATE [3] or UIMA [4]). NLP services can be added and removed from here as required by a language engineer.

Tier 4: Resources. The final tier "Resources" contains the descriptions of the language services using an OWL-DL ontology. Furthermore, indexed documents as well as external documents (e.g., on the Web) count as resources.

3.2 Implementation

We now discuss some selected aspects of the current implementation of our architecture and briefly describe the process of integrating new (desktop) clients.

Language Service Description and Management. We start discussing the implementation from the status quo, a common component-based NLP framework—in our case, GATE [3]. The NLP subsystem allows us to load existing language services, provide them with input documents and parameters, run them, and access their results. The GATE framework's API also allows to access all the artifacts involved in this process (documents, grammars, lexicons, ontologies, etc.). Language services take the form of (persistent) *pipelines* or *applications* in GATE, which are composed of several sequential *processing resources* or *PRs*.

To describe the deployed language analysis services, we developed a comprehensive OWL ontology[3] that models all parameters required for finding applicable services, based on the user's task and language capabilities. This ontology is queried by the NLP Service Connector using SPARQL[4] in order to present applicable Semantic Assistants to the user. For example, we can query the service description metadata for all language pipelines that process input documents in German and deliver results in English. Additionally, the ontology contains all information needed to find, load, and execute a language service (including necessary parametrization), and to locate and retrieve the result(s) delivered.

Web Services. Thus far, we can search, load, parametrize, and execute language services. However, all input/output channels are still local to the context of the NLP framework's process. To make NLP services available in a distributed environment, we have to add network capabilities, which we achieve using *Web services*, a standard defined by the W3C:[5] *"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL[6]). Other systems interact with the Web service in a manner prescribed by its description using SOAP[7] messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."* In essence, a requester agent has to know the description of a Web service to know how to communicate with it, or, more accurately, with the provider agent implementing this Web Service. It can then start to exchange SOAP messages with it in order to make use of the functionality offered by the service. Provider agents are also referred to as Web service *endpoints*. Endpoints are referenceable resources to which Web service messages can be sent. Within our architecture (Fig. 1), the central piece delivering functionality from the NLP framework as a Web service

[3] OWL Web Ontology Language Guide, http://www.w3.org/TR/owl-guide/
[4] SPARQL Query Language for RDF, see http://www.w3.org/TR/rdf-sparql-query/
[5] Web Services Architecture, see http://www.w3.org/TR/ws-arch/
[6] Web Services Description Language (WSDL), see http://www.w3.org/TR/wsdl
[7] Simple Object Access Protocol, see http://www.w3.org/TR/soap/

endpoint is the NLP Service Connector. Our implementation makes use of the Web service code generation tools that are part of the Java 6 SDK and the Java API for XML-Based Web services (JAX-WS).[8]

The Client-Side Abstraction Layer (CSAL). We have just published a Web service endpoint, which means that the server of our architecture is in place. On the client side, our *client-side abstraction layer* (CSAL) is responsible for the communication with the server. This CSAL offers the necessary functionality for clients to detect and invoke brokered language services. The implementation essentially provides a proxy object (of class `SemanticServiceBroker`), through which a client can transparently call Web services. A code example, where an application obtains such a proxy object and invokes the `getAvailableServices` method on it to find available language analysis services, is shown below:

```
// Create a factory object
SemanticServiceBrokerService service = new SemanticServiceBrokerService();
// Get a proxy object, which locally represents the service endpoint (= port)
SemanticServiceBroker broker = service.getSemanticServiceBrokerPort();
// Proxy object is ready to use. Get a list of available language services.
ServiceInfoForClientArray sia = broker.getAvailableServices();
```

Client Integration. After describing the individual parts of our architecture's implementation, we now show how they interact from the point of view of a system integrator adding NLP services to a client application. The technical details depend on the client's implementation language: If it is implemented in Java (or offers a Java plug-in framework), it can be connected to our architecture simply by importing the CSAL archive, creating a `SemanticServiceBrokerService` factory, and calling Web services through a generated proxy object. After these steps, a Java-enabled client application can ask for a list of available language services, as well as invoke a selected service. The code example shown above demonstrates that the client developer can quite easily integrate his application with our architecture, and does not have to worry about performing remote procedure calls or writing network code.

A client application developer who cannot use the CSAL Java archive still has access to the WSDL description of our Web service. If there are automatic client code generation tools available for the programming language of his choice, the developer can use these to create CSAL-like code, which can then be integrated into or imported by his application.

3.3 Application Example: Word Processing Integration

Word processor applications are one of the primary tools of choice for many users when it comes to creating or editing content. Thus, they are an obvious candidate for our architecture, bringing advanced NLP support directly to end users in form of "Semantic Assistants." We selected the open source OpenOffice.org[9]

[8] Java API for XML-Based Web Services (JAX-WS), see https://jax-ws.dev.java.net/
[9] Open source office suite OpenOffice.org, see http://www.openoffice.org/

application *Writer* for integration. We developed a plug-in for *Writer* that seamlessly integrates NLP services, like summarization, index generation, or question-answering, into the GUI [8]. These services are discovered using their OWL description and presented to the user in additional dialog windows; the services can be executed on either the current document (e.g., for summarization) or a (highlighted) selection (e.g., for question-answering). Results are delivered depending on the produced output format, e.g., as a new *Writer* document or by opening a Web browser window.

4 Conclusions

In this paper, we described the design, implementation, and deployment of a service-oriented architecture that integrates end-user (desktop) clients and natural language processing frameworks. Existing applications, or newly developed ones, can be enhanced by so-called Semantic Assistants mediated through our architecture, e.g., information extracting systems, summarizers, and sophisticated question-answering systems. Existing NLP services can be added to the architecture without the need to change any code, thereby immediately becoming available to any connected client.

Our work fills an important gap in the emerging research area intersecting NLP and software engineering that has been neglected by existing approaches. The developed architecture [9] will be made available under an open source license, which we hope will foster a vibrant ecosphere of client plug-ins that finally allow to bring the hard won advances in NLP research to a mass audience.

References

1. Feldman, R., Sanger, J.: The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data. Cambridge University Press (2006)
2. Rhodes, B.J., Maes, P.: Just-in-time Information Retrieval Agents. IBM Syst. J. **39**(3–4) (2000) 685–704
3. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In: Proc. of the 40th Anniversary Meeting of the ACL. (2002) http://gate.ac.uk.
4. Ferrucci, D., Lally, A.: UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. Natural Language Engineering **10**(3-4) (2004) 327–348
5. Corney, D.P., Buxton, B.F., Langdon, W.B., Jones, D.T.: BioRAT: Extracting Biological Information from Full-Length Papers. Bioinformatics **20**(17) (November 2004) 3206–3213
6. Cerbah, F., Daille, B.: A Service Oriented Architecture for Adaptable Terminology Acquisition. In: Proc. NLDB. (2007)
7. Witte, R., Gitzinger, T.: Connecting Wikis and Natural Language Processing Systems. In: Proc. of the 2007 Intl. Symp. on Wikis (WikiSym 2007). (2007)
8. Gitzinger, T., Witte, R.: Enhancing the OpenOffice.org Word Processor with Natural Language Processing Capabilities. In: Natural Language Processing resources, algorithms and tools for authoring aids, Marrakech, Morocco (June 1 2008)
9. Semantic Assistants Architecture. http://semanticsoftware.info