Enhancing the OpenOffice.org Word Processor with Natural Language Processing Capabilities

Thomas Gitzinger[1] **and René Witte**[2]

[1]Institut für Programmstrukturen und Datenorganisation (IPD)
Universität Karlsruhe (TH), Germany

[2]Department of Computer Science and Software Engineering
Concordia University, Montréal, Canada
http://semanticsoftware.info

Abstract

Today's knowledger workers are often overwhelmed by the vast amount of readily available natural language documents that are potentially relevant for a given task. Natural language processing (NLP) and text mining techniques can deliver automated analysis support, but they are often not integrated into commonly used desktop clients, such as word processors. We present a plug-in for the OpenOffice.org word processor *Writer* that allows to access any kind of NLP analysis service mediated through a service-oriented architecture. *Semantic Assistants* can now provide services such as information extraction, question-answering, index generation, or automatic summarization directly within an end user's application.

1. Introduction

Information Retrieval (IR) is, in some respect, a solved problem: Users nowadays have immediate access to vast amounts of information. Popular search engines, such as *Google*, can deliver more documents in a fraction of a second than any human can process in a lifetime. As (Simon, 1971) pointed out, "A wealth of information creates a poverty of attention," and this statement certainly fits the information age. Practically no one dealing with large amounts of reports, literature, drafts, articles and the like, can read everything they would like to. This becomes a problem when decisions have to be made and not all the information that is theoretically available can be taken into account. It becomes a problem when an expert or a reporter must gain an overview of a large corpus of literature, be it news articles, opinion pieces or technical reports, and has a very limited time frame.

While one might argue that retrieval speed and precision of IR can still be improved, we believe the most important advances in the near future will focus on improving the automatic processing of retrieved information, thereby allowing the human user to gain time for his actual task of evaluating the information and creating new value from it. But although natural language processing (NLP) and text mining research has made impressive progress over the last decade, the developed technologies have not yet found wide adoption in end-user tools commonly used for reading and developing content. Knowledge workers[1] in particular could make immediate use of a wide selection of NLP services, such as summarization, index generation, or question-answering, if they just had access to them from their desktop tools, like email clients, Web browsers, or word processors.

In this paper, we present a strategy for integrating any kind of NLP analysis service into a word processing application, the OpenOffice.org[2] *Writer* program. Based on an existing service-oriented architecture, a plug-in created for *Writer* allows to dynamically find, parametrize, and execute language services. That is, in this work we are not concerned with the development of new NLP services, but rather investigate how *any* existing service can be integrated into an end user's tool. A simplified overview of this idea is shown in Figure 1.

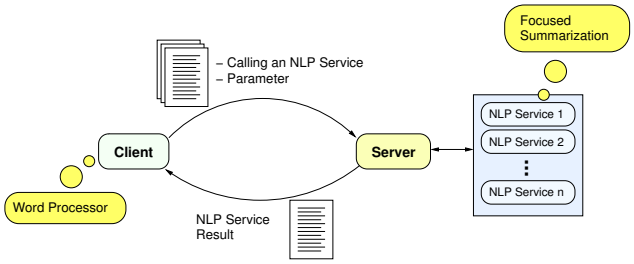

Figure 1: Invoking NLP services directly from a user's client

Our approach allows for a complete reversal of common knowledge acquisition processes: whereas today's knowledge worker has to leave his text processing application to search for relevant information (e.g., through *Google*), process the retrieved results manually, and then continue working on his task, we propose to integrate knowledge retrieval, analysis, and content development into the end-user client. Thus, a user does not have to interrupt his workflow but rather relies on external *Semantic Assistants*, which provide NLP analysis services called directly from the end user's word processing application. These assistants can, for example, search for relevant information on the Web and produce a multi-document summary of the knowledge relevant for the user's current context.

This paper is structured as follows: In the next section,

[1]This term was coined by P. Drucker in 1959. Also known as intellectual worker or brain worker, it emphasizes a worker's capability to work as an expert in a subject matter, rather than, say, through physical labor. See, e.g., http://en.wikipedia.org/wiki/Knowledge_worker.

[2]OpenOffice.org, http://openoffice.org

we describe some application scenarios relevant for our approach in more detail. The design and implementation of our NLP/word processing integration is covered in Section 3. Example applications of the developed solution are presented in Section 4. Related work is discussed in Section 5. Finally, conclusions are presented in Section 6.

2. Application Scenarios

In this section, we describe a number of application scenarios to further motivate our approach of integrating NLP capabilities into a word processor. These scenarios are meant to illustrate the everyday problems that people face in numerous professions and activities. The actors in these scenarios are all knowledge workers, meaning that they have to find, identify, evaluate, and incorporate considerable amounts of existing knowledge in order to do their work.

Scenario 1: Authoring and Analyzing Documents. Editors and journalists continuously face deadlines for delivering articles. Although relevant knowledge—such as previous articles, web pages, and research papers—are readily available online, the typically vast number of hits delivered by an Internet or desktop search makes it impossible to review all documents manually. In this scenario, *focused* multi-document summarization can help the user by presenting an extract of information relevant for the task at hand. The generation of this kind of summary has been investigated for several years within the *Document Understanding Conference* (DUC)[3] competition and lends itself very well to an integration into an office tool: the user can simply highlight a text segment containing questions or other pertinent context information (usually between one and several sentences[4]), which is then used to find relevant documents with an IR system (e.g., from the Web, a digital library, or a local document repository). The summarizer can then prepare a focused multi-document summary of these documents, which contains only those pieces of information relevant for the context question(s). This processing can be performed in the background after the user requested the summary, thereby allowing him to continue work on other parts of his document.
This idea can be further enhanced by adding machine translation tools into the NLP processing pipeline (see the DUC tasks on cross-language summarization), thereby allowing end users access to knowledge in languages they do not speak themselves.

Scenario 2: Information Extraction. Often, knowledge workers require only particular information from a set of documents while working on a task. This might simply be a list of *person* or *company* names or entity relation information, e.g., between *persons* and *events*. This can be achieved with off-the-shelf information extraction (IE) tools, such as the ANNIE system delivered with the GATE framework (Cunningham et al., 2002). Additionally, domain-specific IE can support users in specific knowledge management tasks: For example, a biomedical researcher might need a list of all mutated *proteins* from a set of papers (Witte et al., 2007); and a software engineer might need to find all *method* names covered in a system's documentation (Witte et al., 2008). By integrating IE services, the user can either opt to extract the information from a document he his currently working on, or, like in the previous scenario, on a set of external documents.

Scenario 3: Index Generation. Adding a classical book index to a document is often tedious work, especially when it has not been planned from the start. Simple NLP pipelines can facilitate this task—for example, by performing noun phrase (NP) chunking and building an inverted index from the head noun (first level) and modifier (second level) slots. Using the information extraction techniques mentioned above, specialized indices can additionally be generated, such as person or organization names. Integrated into the word processing application, a draft index can be created directly within the document window, and then further edited and refined by the user.
Index generation can also be applied to *external* documents: instead of looking at text summaries of potentially relevant documents for the current task, a user can also request the creation of a book-type index from documents retrieved through an IR engine, using this as a further navigational aid.

Other NLP Services. The scenarios here are by no means exhaustive—they simply illustrate how enormously useful standard NLP techniques that are already available today can become for an end user when integrated into a standard desktop tool. We expect that future NLP analysis services will be designed directly for deployment in a service-oriented architecture and thereby target the needs of end users even better.

3. Design and Implementation

We now present our solution to the integration of word processing and NLP services. In the first subsection, we briefly describe our service-oriented architecture for NLP/client integration. The second subsection then describes in detail our integration of the OpenOffice.org *Writer* word processor into this architecture. The service-oriented approach was chosen for its ability to model NLP services at a high level of abstraction, thereby hiding the technical aspects of their implementation from the end-user clients. To allow the recommendation of NLP services based on the user's context, i.e., his language capabilities, current task, and client, we provide an ontology model that connects these aspects with available language services.

3.1. Service-Oriented System Architecture

To facilitate the integration of end-user (desktop) clients with natural language processing services, we developed a service-oriented architecture (SOA) that allows to easily connect arbitrary clients with an NLP framework using W3C Web Services.[5] The design and implementation of this architecture is described in (Witte and Gitzinger, 2008). Here, we

[3] Document Understanding Conference, http://duc.nist.gov

[4] An example for such a context is (from DUC 2005): *"What countries are or have been involved in land or water boundary disputes with each other over oil resources or exploration? How have disputes been resolved, or towards what kind of resolution are the countries moving? What other factors affect the disputes?"*

[5] W3C Web Services, http://www.w3.org/TR/ws-arch/

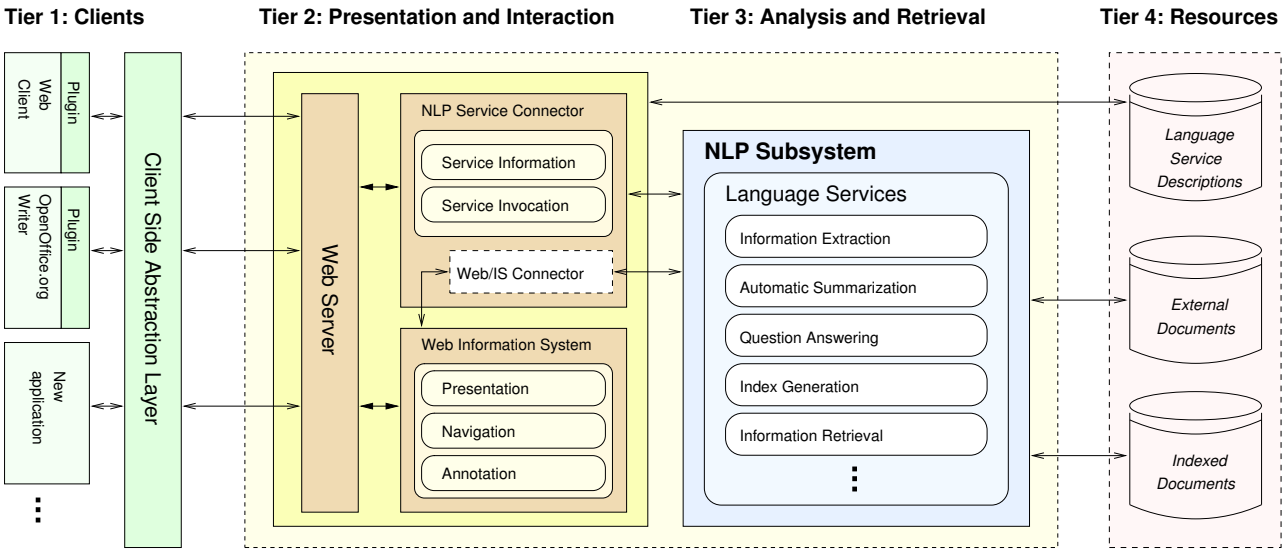

Figure 2: The *Semantic Assistants* architecture for integrating text analysis services and end-user clients

only provide a brief overview to illustrate the steps needed for integrating a new client, in this case, a word processor.
An overview of the architecture is shown in Figure 2. It is based on a typical multi-tier information system design. Clients (Tier 1) provide access to an end-user. Here, we only discuss the integration of a word processor, namely the OpenOffice.org *Writer* application, but our architecture has been designed to allow connections from any kind of client. A client-side abstraction layer (CSAL) provided by our architecture contains a number of pre-defined methods commonly needed for integrating NLP into end-user clients, such as finding available services and handling input/result format conversions. Tier 2 is concerned with handling the interaction between the clients and the NLP frameworks. Since we rely on Web Services for the communication, a web server handles the management of service discovery and invocation, based on WSDL[6] descriptions and SOAP[7] messages. To connect with a concrete NLP framework, our architecture contains an "NLP Service Connector," which is also part of Tier 2. The current version of our architecture supports the GATE framework (Cunningham et al., 2002) for NLP service execution, which forms Tier 3 of our architecture (other frameworks, such as UIMA (Ferrucci and Lally, 2004), can be integrated in the future). Resources form Tier 4, which includes documents stored locally or on a network (including the Internet), as well as metadata about available NLP services, which are formally described using an OWL-DL[8] ontology.
To integrate a new client, the following four steps have to be performed:

1. From the client application, import the Java archive containing our implementation of the client-side abstraction layer (CSAL).

2. If necessary, tell the CSAL the address of the Web service endpoint. The CSAL classes that need to know the address have a default value for it.

3. Create a `SemanticServiceBrokerService` object, which serves as a factory for proxy objects.

4. Create such a proxy object. This is your "remote control" to the Web service. You can call all methods that have been published through the Web service on this object.

We can now discuss how these steps are performed for a concrete client, a word processor, to integrate it into our architecture.

3.2. The OpenOffice.org Writer Plug-in

The OpenOffice.org application suite offers a mechanism to add application extensions, or *plug-ins*. We used this mechanism to integrate OpenOffice.org's word processing application *Writer* with our architecture, and thus equip the *Writer* with Semantic Assistants (Figure 3).
Our primary goal for the *Writer* extension was to be able to perform text analysis on the current document. This text can, for instance, be a large document from which information should be extracted, or a problem statement consisting of a few questions, which serves as input for a question-answering (QA) Semantic Assistant. Especially for the last use case, it must allow a user to highlight part of a document (e.g., a question) and be able to pass only the highlighted part as input to a language service. Furthermore, the extension must offer the possibility to specify parameters that need to be passed to a selected NLP service.
An OpenOffice.org plug-in is basically a zip file with specific contents and certain descriptions of these contents. The files and directories contained in our zip file are shown in Figure 4. Every plug-in has to include a *META-INF* directory, which contains a file called *manifest.xml*. This XML file lists the elements that come with this plug-in; The concrete manifest file for our plug-in is listed in Figure 5. We can see that it defines three *file-entry* elements specifying the type and location of the following files:

[6]WS Description Language, http://www.w3.org/TR/wsdl
[7]Simple Object Access Protocol, http://www.w3.org/TR/soap/
[8]OWL Web Ontology, http://www.w3.org/TR/owl-guide/

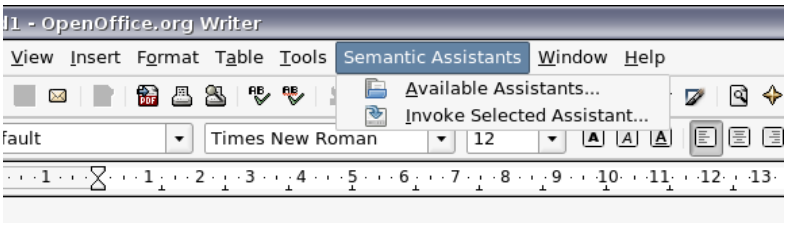

Figure 3: The new "Semantic Assistants" menu entry in OpenOffice.org Writer allows to find and execute NLP services

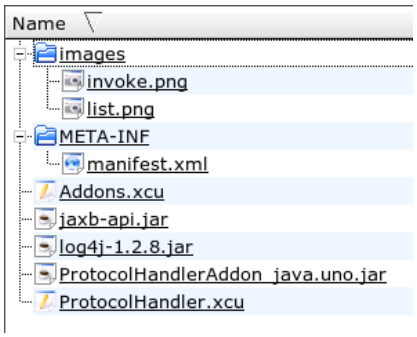

Figure 4: The plug-in file structure

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE manifest:manifest PUBLIC
"-//OpenOffice.org//DTD Manifest 1.0//EN" "Manifest.dtd">
<manifest:manifest
 xmlns:manifest="http://openoffice.org/2001/manifest">
  <manifest:file-entry
    manifest:media-type=
      "application/vnd.sun.star.configuration-data"
    manifest:full-path="Addons.xcu"/>
  <manifest:file-entry
    manifest:media-type=
      "application/vnd.sun.star.configuration-data"
    manifest:full-path="ProtocolHandler.xcu"/>
  <manifest:file-entry
    manifest:media-type=
      "application/vnd.sun.star.uno-component;type=Java"
    manifest:full-path=
      "ProtocolHandlerAddon_java.uno.jar"/>
</manifest:manifest>
```

Figure 5: The *manifest.xml* file for our plug-in

Addons.xcu. This XML file defines how the plug-in should be integrated with OpenOffice.org. In our case, it contains a menu definition, specifying that the menu should only appear in the *Writer* application. For each menu item, we specify which messages should be broadcast throughout the OpenOffice.org runtime system when the menu item is activated.

ProtocolHandler.xcu. This XML file specifies that the messages defined in *Addons.xcu* should be handled by an object of a certain class. This class is provided in the Java archive and must adhere to a certain interface.

ProtocolHandlerAddon_java.uno.jar. This Java archive contains the actual functionality of the plug-in. It holds classes responsible for receiving the messages generated by the menu items, as well as classes responsible for the interaction with the client-side abstraction layer.

Our plug-in creates a new menu entry "Semantic Assistants," as shown in Figure 3. In this menu, the user can inquire about available services, which are selected based on the client (here *Writer*) and the language capabilities of the deployed NLP services (described in service metadata). The dynamically generated list of available services is then presented to the user, together with a brief description, in a separate window, as shown in Figure 6. Note that the integration of a new service does not require any changes on the client side—any new NLP service created and deployed by a language engineer is dynamically discovered through its OWL metadata maintained by the architecture and so becomes immediately available to any connected client.

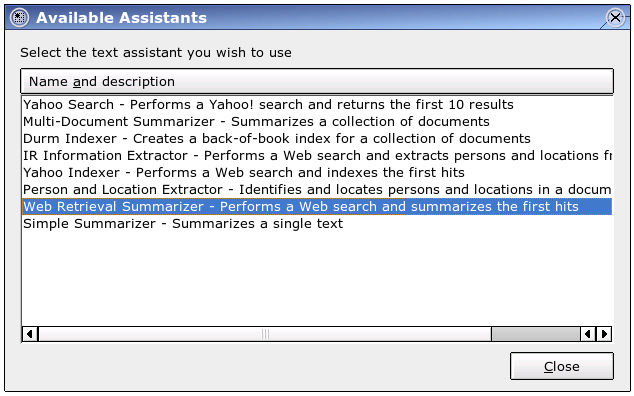

Figure 6: List of available semantic assistants

The user can now select an assistant and execute it. In case the service requires additional parameters, such as the length of a summary to be generated, they are detected by our architecture through the OWL-based service description and requested from the user through an additional dialog window. An example, for the *Web Retrieval Summarizer* assistant, is shown in Figure 7.

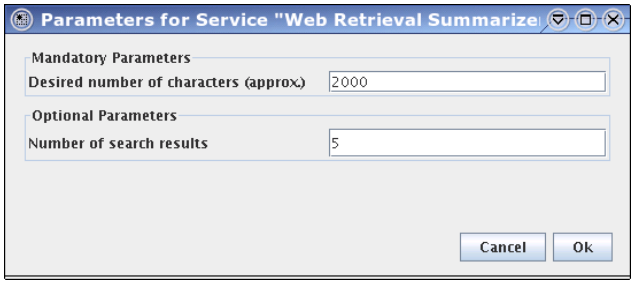

Figure 7: The parameters dialog, which appears when a Semantic Assistant requiring further input is invoked

Once requested, the language service is executed asyn-

chronously by our architecture, allowing the user to continue his work (he can even execute additional services). The sequence diagram in Figure 8 shows the execution of a service through the various tiers described in Section 3.1. Note that all low-level details of handling language services, such as metadata lookup, parametrization, and result handling, are hidden from the client plug-in through our client-side abstraction layer.

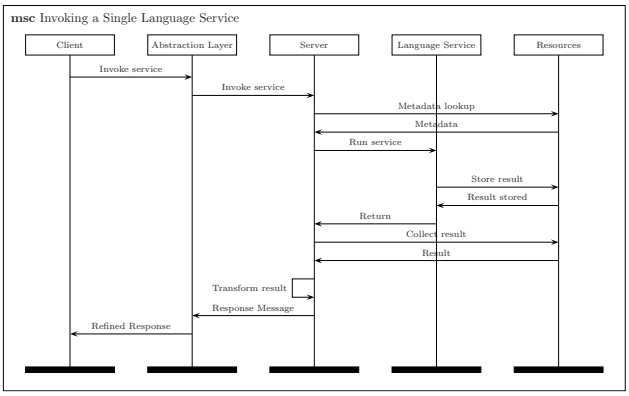

Figure 8: The client invokes a language service and receives a result

4. Application

One direct use case of our Semantic Assistants is to satisfy information needs of a knowledge worker. As motivated in the introduction, language services can deliver focused analysis results directly within the client—here a word processor—needed to perform a task, rather than interrupting the user's workflow by forcing him to perform an external (Web) search.

For example, a user might develop a report on the global climate change and needs information on the role of *"DMSP in the Atlantic marine biology."*[9] With our OpenOffice.org plug-in, the user can simply highlight this phrase in the *Writer* editor window and select the "Web Retrieval Summarizer." This is a compound Semantic Assistant, which performs two functions: In a first step, a selected number of hits from a Yahoo! search using the highlighted phrase is retrieved to build a corpus on-the-fly. This corpus is then fed into the context-sensitive multi-document summarizer ERSS (Witte and Bergler, 2007) to produce a summary. All these actions are performed in the background, allowing the user to continue with other parts of his report. When the summary is ready, the architecture notifies the plug-in, which then presents the generated summary in a new window, as shown in Figure 9. The user can now analyze, edit, or copy parts of the summary within his workflow.

Note that our architecture does not require that the user's word processor and the NLP service reside on the same physical machine. Although this is a possible configuration for personal knowledge management, the underlying Web Service framework also allows to access specific analysis tools from an external service provider. For example, a university might want to deliver question-answering services targeted to its students, answering questions about courses and facilities. A commercial scientific publisher might want to offer a "related work finder" analysis service, similar to the one presented by (Zeni et al., 2007), to scientists writing research papers or proposals.

[9] DMSP stands for "Dimethylsulfoniopropionate" and is a component of the organic sulfur cycle.

5. Related Work

There is not much previous work that deals with the software engineering aspects of integrating NLP services into existing desktop tools.

In their article "Just-in-time information retrieval agents" (JITIR agents), (Rhodes and Maes, 2000) presented a plug-in for the Emacs text editor called the Remembrance Agent (RA). The RA presents, in a special sub-window at the bottom of the Emacs window, a small list of documents that are related to the document currently being written or read. These suggestions can come from multiple different databases or e-mail archives, and are periodically updated. More concretely, every few seconds, an Information Retrieval (IR) process is triggered, performing a search on the specified databases based on, for example, the last 500 words written or the e-mail message that has just been opened for viewing. The results of this IR process are then ranked and listed in the sub-window at the bottom. The second JITIR agent presented in the article, Margin Notes, works by the same principle as the Remembrance Agent. It rewrites displayed Web pages and adds annotations (e.g., links to related e-mail documents) in a separate column on the right to the Web page.

(Colbath and Kubala, 2003) presented TAP-XL, an "automated analyst's assistant." The system's front end is an extension to Microsoft Word. The user writes an initial problem statement, which the system analyzes and uses to retrieve possibly related articles or documents from Internet sources. Unlike with the Remembrance Agent mentioned above, which performs its work in the background without the user actively triggering that work, the user's interaction with the TAP-XL system is more conscious, as he actively poses a problem statement that the system processes. The processing elements that make TAP-XL work form a distributed system of Web Services. Among these services are machine translation, document clustering, multi-document summarization, and fact extraction. The results of these components are stored in a central repository, from where they are accessible to downstream technologies like the word processing front end. The documents to feed this whole system come from a commercial source, as well as from Web harvesting.

These approaches differ from our work in that they are strictly bound to one field of application (e.g., word processing for TAP-XL). By providing an open, client-server, standards-based infrastructure, we can bring NLP to the end user practically regardless of what kind of application she is using. Moreover, the mentioned text assistants' functionality is confined to providing possibly relevant documents. In contrast to that, we want to offer a theoretically open-ended number of NLP services, including machine translation, information extraction, automatic summarization, and automatic indexing. While referenced applications like TAP-XL

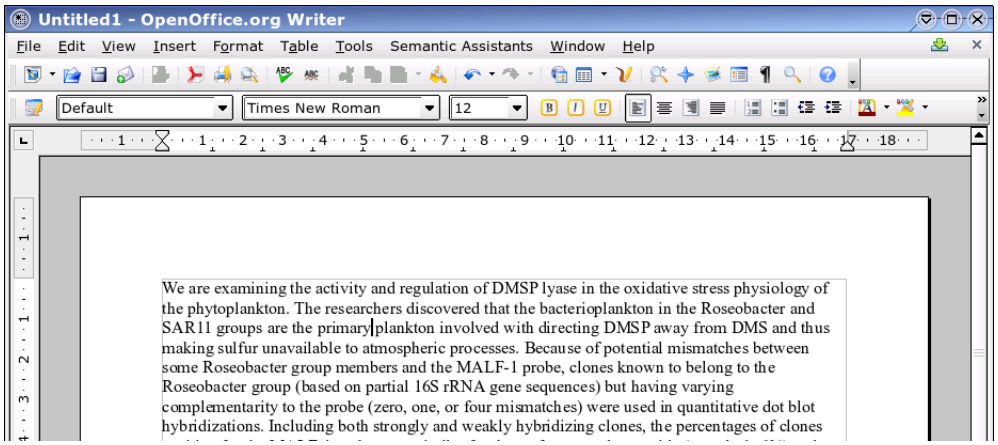

Figure 9: The result of the "Web Retrieval Summarizer" Semantic Assistant, answering the user's question, is displayed as a new document

have their NLP functionality largely built right into them, we follow a different approach in that we clearly distinguish between service requester (the client), the requested service (semantic text assistant), and the underlying architecture. The services offered as part of our architecture are entirely self-contained and unaware of anything that is going on "on the outside." On the other end of the communication, the client program is initially unaware what semantic services it can make use of. It only knows how to talk to the Web Service connecting the two ends. This separation simplifies client development, bears no influence on the NLP development, and allows for higher flexibility.

6. Summary and Conclusions

Today's knowledge workers face constant "information overload." Many NLP techniques and text mining systems have been developed to address the (semi-)automatic analysis of natural language texts—but none of these are of any use to an end user if they are not readily accessible within the applications commonly found on today's desktop environments. In this paper, we described a novel solution for the integration of any kind of NLP service into a word processor application. Our underlying architecture together with the created plug-in perform the "heavy lifting," so to speak, of software engineering work necessary to bring semantic analysis tools to end users. Through its service-oriented approach, it decouples the creation of NLP pipelines and their access from connected clients, so that NLP developers do not need to be concerned with the technical details on how their created services will become available within desktop applications, and client developers likewise do not need to know about the intricacies of developing natural language processing pipelines, since all they see are Web Service descriptions of these services. The developed architecture has been developed based on open standards and open source tools and will also be made available as free software.

Further improvements on the plug-in side will focus on enhanced formatting of NLP results, which can come in diverse formats (e.g., unstructured summaries, structured tables, XML or OWL files). Further plug-ins extending other end-user clients with NLP services are under investigation, like for an email client, Web browser, and software development environment.

On the architectural side, a more detailed handling of the user's current context using our ontological model will allow for a more fine-grained pre-selection and -parametrization of available language services. Together with automated reasoning capabilities of OWL-DL ontology reasoners, an agent-like approach becomes possible, where relevant services are not only executed explicitly, but can also be automatically scheduled by the architecture based on the user's current behavior.

7. References

Sean Colbath and Francis Kubala. 2003. TAP-XL: An Automated Analyst's Assistant. In *NAACL '03: Proc. of the 2003 Conference of the North American Chapter of the ACL*, pages 7–8.

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proc. of the 40th Anniversary Meeting of the ACL*. http://gate.ac.uk.

D. Ferrucci and A. Lally. 2004. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*, 10(3-4):327–348.

B. J. Rhodes and P. Maes. 2000. Just-in-time Information Retrieval Agents. *IBM Syst. J.*, 39(3–4):685–704.

Herbert A. Simon. 1971. Designing Organizations for an Information-Rich World. In *Computers, Communication, and the Public Interest*, pages 40–41. The John Hopkins Press.

René Witte and Sabine Bergler. 2007. Fuzzy Clustering for Topic Analysis and Summarization of Document Collections. In *Proc. Canadian A.I. 2007*, LNAI 4509, pages 476–488, Montréal, Québec, Canada, May 28–30. Springer.

René Witte and Thomas Gitzinger. 2008. A General Architecture for Connecting NLP Frameworks and Desktop Clients using Web Services. In *NLDB 2008*, LNCS. Springer.

René Witte, Thomas Kappler, and Christopher J. O. Baker. 2007. Enhanced Semantic Access to the Protein Engineering Literature using Ontologies Populated by Text Mining. *Int. Journal of Bioinformatics Research and Applications (IJBRA)*, 3(3).

R. Witte, Q. Li, Y. Zhang, and J. Rilling. 2008. Text mining and software engineering: an integrated source code and document analysis approach. *IET Software*, 2(1):3–16.

N. Zeni, N. Kiyavitskaya, L. Mich, J. Mylopoulos, and J.R. Cordy. 2007. A Lightweight Approach to Semantic Annotation of Research Papers. In *Proc. NLDB*.