SOA implementation of the eHumanities Desktop

Rüdiger Gleim, Alexander Mehler, Alexandra Ernst

Goethe-University Frankfurt
Senckenberganlage 31, D-60325 Frankfurt, Germany
gleim@em.uni-frankfurt.de, mehler@em.uni-frankfurt.de, ernst.alexandra@gmail.com

Abstract

The eHumanities Desktop is a system which allows users to upload, organize and share resources using a web interface. Furthermore resources can be processed, annotated and analyzed in various ways. Registered users can organize themselves in groups and collaboratively work on their data. The eHumanities Desktop is platform independent and runs in a web browser. This paper presents the system focusing on its service orientation and process management.

Keywords: Digital Humanities, Web Services, Asynchronous Processes, TEI P5

1. Introduction

Developments in text-technology and digital humanities in recent years document a strong shift towards web services and applications. The diversity of research interests is reflected by the rich applications, platforms and frameworks being developed. Textgrid (Kerzel et al., 2009) is a research association which aims at supporting access and exchange of resources in the humanities. Textgrid provides the Textgrid Lab[1] as a platform to work on resources. The project eAqua[2] (Büchler et al., 2010) aims at knowledge extraction based on processed antique sources and offers classical scholars convenient means to work on the results.
Furthermore, digital editions of works offer additional value by allowing users to browse and query meta data and to perform full text search on resources. For example, the *Heinrich Heine Portal*[3] provides a digital edition of the works of Heinrich Heine (Füllner and Fournier, 2002). The letters of Vincent van Gogh have also been published as a digital edition (Jansen et al., 2010). *Inscriptiones Graecae*[4] is a project at the Berlin-Brandenburgische Akademie der Wissenschaften which provides access to Greek inscriptions. In addition, there are many more projects not explicitly named here. Research associations, notably CLARIN[5] (Tamás Váradi and Koskenniemi, 2008) and DARIAH[6] (DARIAH, 2010), focus on creating common infrastructures and improving exchange and interoperability of services.
Desktop applications continue to do well in specific domains. Web applications on the other hand are typically not designed as mere web portations, but provide additional value. Some principal advantages are the ease or elimination of software installation as well as platform independence. But the true benefits are the means to share and use resources *and* services online. Web applications and services thus allow for collaborative work on research artifacts in a way which would hardly be possible otherwise.
The concept of *Service Oriented Architectures* (SOAs) has been proposed as a means to integrate isolated services in order to provide more sophisticated functionality. The general idea of SOAs is described as the orchestration of services rather than the service itself. How a SOA is implemented in a specific domain is not standardized, but has to be designed according to the requirements of the respective application area. In this article we describe how the eHumanities Desktop (Gleim and Mehler, 2010) adopts the concept of Service Oriented Architectures to provide a platform for research in the digital humanities.

2. SOA implementation of the eHumanities Desktop

The eHumanities Desktop is a system which allows users to upload, organize and share resources using a web interface. Furthermore resources can be processed, annotated and analyzed in various ways. Registered users can organize themselves in groups and collaboratively work on their data. Figure 1 shows a screenshot of the eHumanities Desktop's web client in action. It is platform independent and runs in a web browser.

2.1. Overview of Architecture

The functionality of the overall system is realized by means of *application modules*. The development of new modules is mainly driven by research projects. Developers can benefit from the eHumanities Desktop framework by using its APIs to manage access permissions, storage of resources and incorporate other modules, applications or remote webservices to perform their task. Thus typical aspects like user- and rights management as well as storage handling are already taken care of by service modules. The developer of

[1] http://www.textgrid.de/1-0.html
[2] http://www.eaqua.net
[3] http://www.hhp.uni-trier.de/Projekte/HHP/start
[4] http://telota.bbaw.de/ig/
[5] http://www.clarin.eu
[6] http://www.dariah.eu

Figure 1: Screenshot of the eHumanities Desktop's Web Client.

a new application can concentrate on the new functionality. An application module typically consists of a *client module* which provides a user interface and a *service processor* on the server which executes incoming commands. In the following section we will describe the system architecture in more detail. We start with an overview that provides a general picture of the components and technologies being used. Then we describe the process of executing a service request in the eHumanities Desktop.

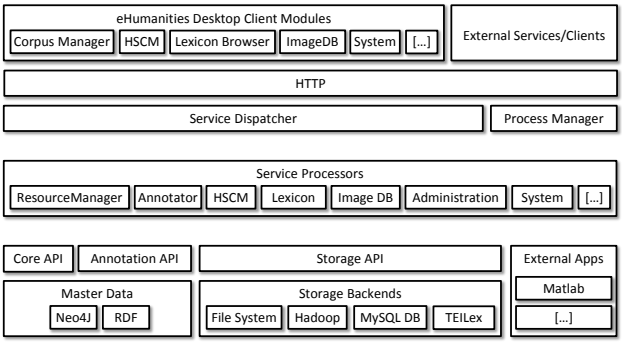

Figure 2: Architecture of the eHumanities Desktop Framework.

Figure 2 gives an overview of the architecture of the eHumanities Desktop. In terms of layers, the eHumanities Desktop consists of the client modules, the service processors and the system APIs. The client modules are widely based on JavaScript using the Ext JS Framework[7] which greatly simplifies development. In addition, some functions also require Java Applets and Flash.

The service processors implement the functionality of the system and are accessible by a web API. They are implemented in Java and run as part of a web servlet on Apache Tomcat. Each processor covers a certain area, for example resource management, annotation, text preprocessing, categorization and the like. In order to accomplish their task, service processors can use the eHumanities Desktop APIs to access the master data, storage and annotion system. In addition they can call local applications on the servers or other web services on the internet. The primary means to access the web API is using the eHumanities Desktop Client. Nonetheless service requests can also be triggered by other means, for example PHP, C++ or Java applications using HTTP Post/Get. Thus the eHumanities Desktop can also be used by other services and applications.

The system APIs give access to resource and user management, annotations and storage. Since we intend to focus on the service orientation of the system we do not describe them in depth here. The repositories, documents, users and resources are managed in a master database. Figure 3 shows the class diagram of the master data. It is currently implemented based on the graph database Neo4J[8] in combination with Lucene[9] for indexing.

The storage API manages access to the document contents. The way a document is stored in the system is not fixed. By implementing an interface, developers can rather add new technologies as required. How a

[7] http://www.sencha.com/products/extjs

[8] http://neo4j.org/

[9] http://lucene.apache.org

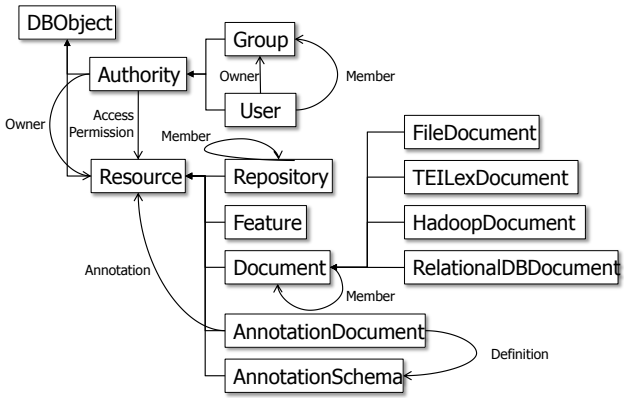

Figure 3: Class diagram of the master data.

new document is stored is decided based on its file format (or MIME-type[10]): For example a TEI P5 document will be stored in the TEILex DB, a binary file in a Hadoop/HDFS[11] cluster and even MySQL databases can be imported as documents. From the user perspective the way a document is stored does not make any difference. The annotation API supports RDF to annotate resources in the system.

2.2. Processing of Service Requests

This section describes the process of retrieving, executing and responding to a service request. Figure 4 illustrates this approach. At the beginning of processing a call, an incoming service request is checked for its authentication. A login command would trigger a new session in the *session handler* and return a unique session key to authenticate subsequent requests. The next step is to parse the request and to determine which service processor is in charge. This is determined by checking the prefix of the incoming command string. A failure in one of these checks will trigger an error message to the client.

Service requests as well as responses are normally formatted in the JavaScript Object Notation Language (JSON)[12] and transmitted via HTTP. This is due to the primary means of accessing the system via the eHumanities Desktop Client which is widely written in JavaScript. As already noted any other means to issue HTTP Get/Post can be used as well.

Next, transactions on the underlying databases are started to ensure that any changes to the system during the processing can be rolled back in case of an error. Then an instance of the proper service processor is created, registered in the *process manager* and started. The process manager is an essential component to keep track of which user runs which processes at a given time. Every process has a process id, name and owner. Furthermore the time when the process was triggered is stored and its current progress is tracked. The web API allows users to list their current processes and cancel them if desired. Canceling a process is interesting for long runnning processes such as complex analyses. However the user can only *trigger* a process to stop. The process has to regularely check whether it should continue or if the user wishes to cancel it. The same applies to updating the current progress – a task only the process itself can answer. A hard stop of a process would be possible but could lead to unpredictable results and data corruption. Therefore this option is currently switched off – only "soft" stopping is supported as described. Figure 5 shows a screenshot of the graphical frontend of the process manager with two indexing tasks running.

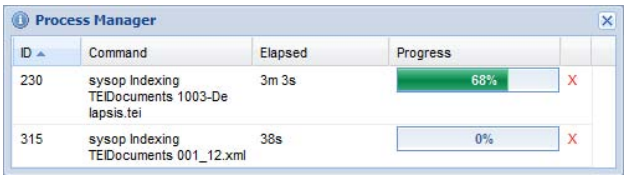

Figure 5: Screen of the Process Manager showing two asynchronous tasks of the user.

The service processor checks permissions of the requesting user on resources which have been passed as parameters. In case of a failure the client is informed accordingly. Then the computation itself is performed. A command processor can use the eHumanities Desktop's APIs to accomplish its task, as for example to import a TEI P5 document into the TEILex database and link its tokens to the lexicon entries. This task can be quite complex and time consuming, depending on the size of the document. For direct feedback to the client the process would take to long and cause a timeout on the client side, which waits for a response. In such cases a developer of a service processor can choose to execute the task in a thread *asynchronously*. This means that the task is executed as a separate thread. After the thread has been triggered control is given back to the system which returns a message to the client that the process as been started. The transactions and resources remain open and the process manager keeps the process as well. The client can track the progress of the processes, safely log out and check for the results later by logging in again. When an asynchronous process has finished it triggers the steps to unregister it from the process manager and commit transactions. These steps are performed automatically when a *synchronous* task was completed. A drawback of HTTP is that the server can not actively inform the client when a process has finished. Thus a response message is only sent for synchronous tasks.

2.3. Use Case: Representing TEI P5 and Lexica with TEILex

To give an example of a typical application module and its components we now describe the *TEILex* subsystem. It is one of the most recent additions to the eHumanities Desktop. It provides a seamless integra-

[10] `http://www.iana.org/assignments/media-types/index.html`

[11] `http://hadoop.apache.org/hdfs/`

[12] for more informations about JSON see `http://json.org`

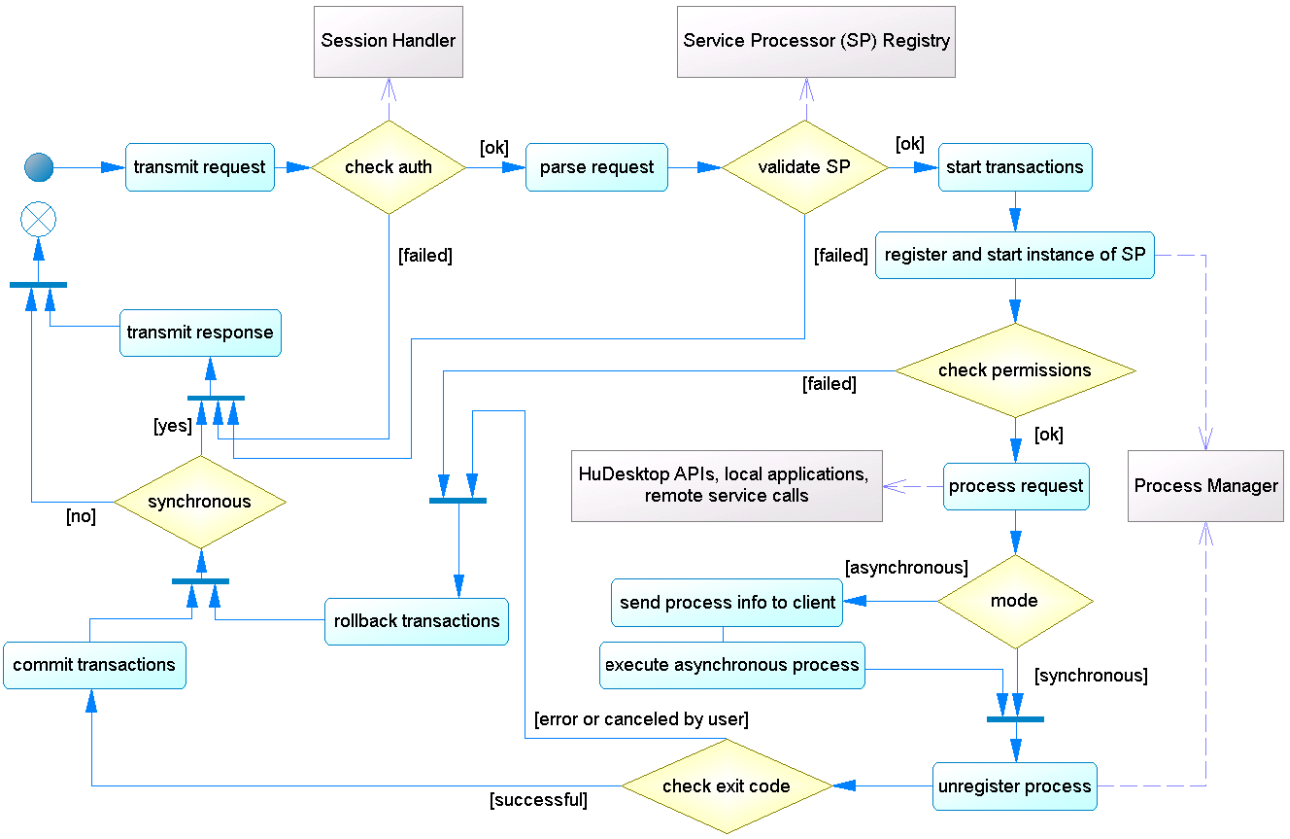

Figure 4: UML Action Diagram of how service requests are processed.

tion of lexica with text collections encoded in TEI P5. This is achieved by linking lexical information of text tokens directly with lexicon entries. The integration of lexicon and TEI representation enables the user to systematically correct erroneous linguistic annotations in preprocessed texts and to perform disambiguation. Figure 6 shows a screenshot of the lexicon browser which exemplifies dynamic linking of word forms between text and a lexicon. The visualization uses color codes to inform the user about ambiguities. Uniquely identified word forms are marked green. In this example the word form “spiritus” is highlighted in yellow to state that multiple lexicon entries of this word form exist, that belong to the same lemma. If the item were orange, it would mark an ambiguity that affects different lemmata.

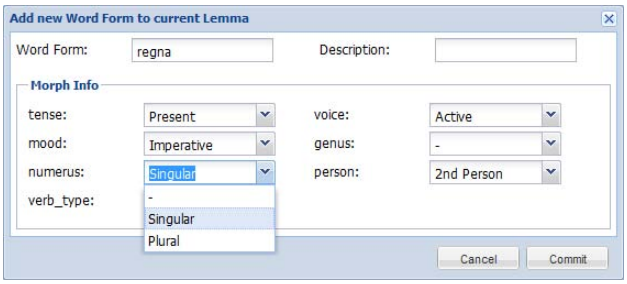

Figure 7: Screenshot of the Lexicon Browser exemplifying the creation of a new word form.

The system allows users to edit and expand lexica as well. Figure 7 exemplifies how a new word form can be created for a previously selected lemma. Both lexicon and text representation are designed to be dynamic and extensible by users. This is a demanding task because highly dynamic databases are more difficult to optimize for efficient reading *and* writing. Since an import may take minutes or even hours, depending on the size of the document set, this task is done asynchronously. A user can trigger the import of a set of documents and log out. Later he or she can log in again and check on the progress by opening the process manager (see figure 5).

The TEILex is a good example of how different service modules can be put together to achieve an advanced functionality in terms of a SOA. A preprocessor module can be used to convert plain text documents into TEI P5 documents which can be imported into the TEILex system. The TEILex API can then be used by service modules to implement services for client modules or other external clients. The *Lexicon Browser* module serves as an interface to the lexica, while another module, called *HSCM* (Historical Semantics Corpus Management), (Jussen et al., 2007; Mehler et al., 2011) offers an interface to query and explore text corpora. Both modules interact on the client side to lookup words or keywords in context. And both use the same mechanisms on the server to accomplish their tasks.

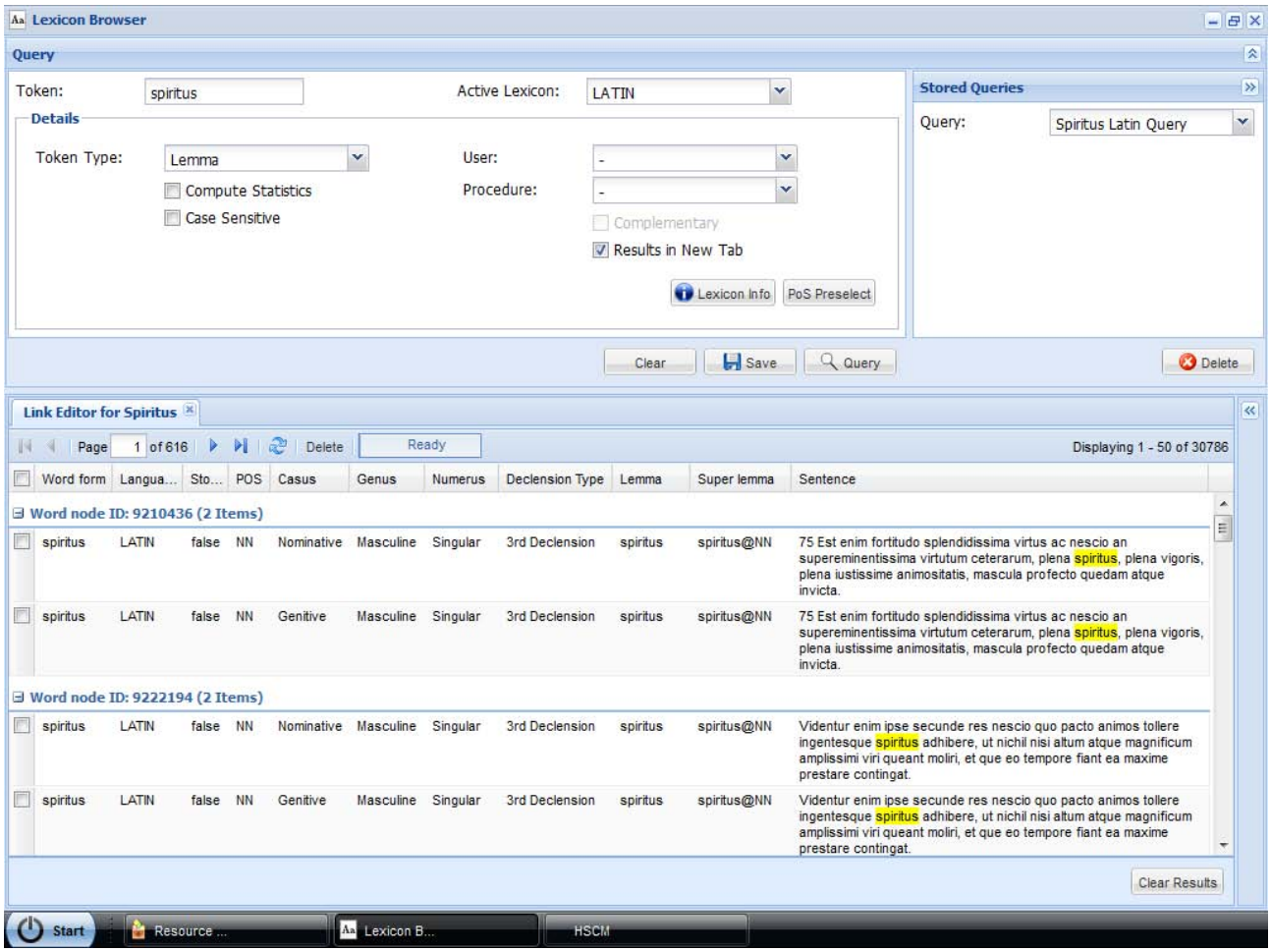

Figure 6: Screenshot of the Lexicon Browser exemplifying dynamic linking of word forms between text and lexicon.

3. Conclusion

We have described the eHumanities Desktop as a service oriented architecture. It supports the development of services related to managing, browsing and analysing resources in the humanities. Special focus was given to the way service requests are handled and the way long running tasks can be managed asynchronously. Current development outside of project specific development is targeting the annotation system and improving the TEILex system. The eHumanities Desktop is open to scientists who are interested in testing and working with the system.

4. Acknowledgements

The eHumanities Desktop core is being developed in the lab for text-technology (computer science) at Goethe University while active development on new services and extensions is done with and by research projects. We gratefully acknowledge the support by the Leibniz project “Politische Sprache im Mittelalter”[13] of Bernhard Jussen and the projects of the LOEWE Schwerpunkt “Digital Humanities”[14], both at Goethe University.

[13] https://semantik.geschichte.uni-frankfurt.de/

[14] http://www.digital-humanities-hessen.de/

5. References

M. Büchler, A. Geßner, G. Heyer, and T. Eckart. 2010. Detection of citations and text reuse on ancient greek texts and its applications in the classical studies: eaqua project. In *Proceedings of the Digital Humanities 2010 Conference*, London, UK.

DARIAH. 2010. Dariah information booklet. http://dariah.eu/index.php?option=com_docman&task=doc_download&gid=383&Itemid=200.

Bernd Füllner and Johannes Fournier. 2002. Das heinrich-heine-portal. ein integriertes informationssystem. In *Beiträge des Internationalen Kolloquiums an der Universität Trier, 8./9. Oktober 2001. Hrsg. von Thomas Burch, Johannes Fournier, Kurt Gärtner u. Andrea Rapp. Trier 2002 (Abhandlungen der Akademie der Wissenschaften und der Literatur, Mainz)*.

Rüdiger Gleim and Alexander Mehler. 2010. Computational linguistics for mere mortals – powerful but easy-to-use linguistic processing for scientists in the humanities. In *Proceedings of the 7th Interna-*

tional Conference on Language Resources and Evaluation (LREC), Valletta/Malta, May 19-21.

Leo Jansen, Hans Luijten, and Nienke Bakker. 2010. Vincent van gogh - the letters. version: December 2010. amsterdam and the hague: Van gogh museum and huygens ing. `http://vangoghletters.org/vg/`.

Bernhard Jussen, Alexander Mehler, and Alexandra Ernst. 2007. A corpus management system for historical semantics. *Sprache und Datenverarbeitung. International Journal for Language Data Processing*, 31(1-2):81–89.

Martina Kerzel, Jens Mittelbach, and Thorsten Vitt. 2009. Textgrid: Virtuelle arbeitsumgebung für die geisteswissenschaften. *KI Künstliche Intelligenz*, (4):36–39.

Alexander Mehler, Silke Schwandt, Rüdiger Gleim, and Bernhard Jussen. 2011. Der eHumanities Desktop als Werkzeug in der historischen Semantik: Funktionsspektrum und Einsatzszenarien. *Journal for Language Technology and Computational Linguistics (JLCL)*, 26(1):97–117.

Peter Wittenburg Martin Wynne Tamás Váradi, Steven Krauwer and Kimmo Koskenniemi. 2008. Clarin: Common language resources and technology infrastructure. In Bente Maegaard Joseph Mariani Jan Odjik Stelios Piperidis Daniel Tapias Nicoletta Calzolari (Conference Chair), Khalid Choukri, editor, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may. European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2008/.