Design and Implementation of Architectures for Interactive Textual Documents Collation Systems

Mohamadou Nassourou
Department of Computer Philology & Modern German Literature
University of Würzburg Am Hubland D - 97074 Würzburg
mohamadou.nassourou@uni-wuerzburg.de

Abstract

One of the main purposes of textual documents collation is to identify a base text or closest witness to the base text, by analyzing and interpreting differences also known as types of changes that might exist between those documents. Based on this fact, it is reasonable to argue that, explicit identification of types of changes such as deletions, additions, transpositions, and mutations should be part of the collation process. The identification could be carried out by an interpretation module after alignment has taken place. Unfortunately existing collation software such as CollateX[1] and Juxta[2]'s collation engine do not have interpretation modules. In fact they implement the Gothenburg model [1] for collation process which does not include an interpretation unit. Currently both CollateX and Juxta's collation engine do not distinguish in their critical apparatus between the types of changes, and do not offer statistics about those changes.

This paper presents a model for both integrated and distributed collation processes that improves the Gothenburg model. The model introduces an interpretation component for computing and distinguishing between the types of changes that documents could have undergone.

Moreover two architectures implementing the model in order to solve the problem of interactive collation are discussed as well. Each architecture uses CollateX library, and provides on the one hand preprocessing functions for transforming input documents into CollateX input format, and on the other hand a post-processing module for enabling interactive collation. Finally simple algorithms for distinguishing between types of changes, and linking collated source documents with the collation results are also introduced.

Keywords: service based software architecture, service brokerage, interactive collation of textual variants, Gothenburg model of collation process

1. Introduction

Textual documents collation is the act of comparing and analyzing several sources of information. CollateX is the successor of Peter Robinson's Collate program that one could use to collate textual documents. However it currently lacks user interfaces for interactively and intuitively carrying out textual documents collation. Juxta is a software tool equipped with user interfaces that offers the possibility for collating and analyzing textual documents. The tool is so far the best visual tool known to me for collating textual documents. However it does not offer users enough means to customize the collation process by setting some pre-processing rules that have to be applied to documents under collation. For instance filtering out or replacing some specific characters is not possible, as well as setting precedence rules so that the collator considers two characters or words as equivalent. Moreover it lacks post-processing interfaces which could allow users to make some adjustments of the critical apparatus, and re-collate the documents.
Among the shortcomings of CollateX and Juxta, there is their inability to offer in their collation results the types of changes and statistics about those changes.
One of the main purposes of documents collation is to identify a base text or closest witness to the base text, by analyzing and interpreting differences also known as types of changes that might exist between those documents.

Since the author of this paper is not a CollateX developer, and it does not offer procedures to achieve the above stated goals, it has been found necessary to extend its functionalities by considering it as a software component whose output could be an input to another system. In this way I improve the Gothenburg model by introducing an interpretation component, which is required to fulfill the purpose of textual documents collation. The interpretation step consists of post-processing collation results in order to differentiate and count the types of alterations that took place in each document. Since CollateX does accept XML as input, an algorithm for linking a collation result to XML documents under collation has also been developed.

This paper presents two architectures for textual documents collation systems that could be used both in integrated as well as distributed environments. The architectures extend CollateX functionalities by introducing an interpretation module necessary for identifying and grouping types of changes in collated documents. Additionally they depict and implement a practical solution for the interactive collation issue. Users have the possibilities through graphical interfaces to iteratively alter collation results and re-collate them.

1. http://collatex.sourceforge.net/
2. http://www.juxtasoftware.org/

2. CollateX

CollateX is a Java library for collating baseless textual documents. Inputs to the library are plain text documents having JSON format. Collation results are objects of aligned tokens arranged in form of matrix that could be visualized as HTML, XML, JSON or any other graphical format. The rows of the matrix are the aligned input documents, and the columns are the aligned tokens of the documents.

It implements the Gothenburg model consisting of tokenization, alignment, and visualization components that are described in [1]. Knowing that, one of the purposes of collating documents (at least from the perspective of religious texts scholars) is to find out the types of changes such as deletions, additions, transpositions, and mutations that occurred in documents, it is therefore essential that CollateX specifically provides this information in order to assist users in interpreting the collation results. Unfortunately at its current state of development it does not.

From my own point of view it would not be sufficient to simply show alignment tables, which do not clearly differentiate between the types of changes that, documents have undergone. Moreover I would suggest having separate interfaces for each type of changes that could allow explicit retrieval of the changes independently.

For instance methods like: getAdditions(), getDeletions(), getTranspositions(), and getMutations() could be offered.

3. Collation Methodology

Textual documents collation refers to the process of measuring proximities between those documents. Proximity consists of similarities and differences that exist among the collated documents. Therefore a collation system is the one that is able to locate common and unshared characters between textual documents with their statistics. Ideally a textual documents collation system should not only look at proximities, but also at actual position of each character in the text. The position of characters is needed in order to identify the types of changes that might have occurred in the documents.

3.1 Definition of Types of Changes

Referred types of changes in this study, are changes that are not necessarily depending on human interpretation such as clarification, overwriting, fixation, and so on. For more information on textual document alterations see [4]. To the contrary these are changes which are identifiable through simple inspection and visualization of a textual document. Among them are:
Deletion which means a witness misses some characters from a base text.
Addition which means a base text misses some characters from a witness text.
Transposition which means the position of some characters present in a witness does not correspond to their counterpart position in a base text.
Mutation which means the positions of two words in base and witness texts are same, but they are different.
There are two types of mutations:

a. Full mutation
   The word of the base text is completely different from that one of the witness
b. Partial mutation
   Part of the altered word from the base text corresponds to that of the witness one

Mutations are more serious than transpositions, because they introduce completely new terms or characters in the documents by deleting and adding at the same location, even if the deleted and added words could be synonymous.

3.2 Types of Textual Documents Collation Systems

There are basically two types of textual documents collation systems:

a. Baseless text based collation
   In the baseless collation process, all the documents (witnesses) are collated against each other. The baseless collation would be appropriate for collating documents with a priori unknown and unpredictable base text. However this approach of collation is unfortunately adding many complications and post processing tasks. It is advantageous to try minimizing the number of documents to collate. Moreover after having determined a base text from the collation result, one would like to determine document(s) that are closest to the base text. The types of changes such as deletions, additions, transpositions, and mutations which are required for the interpretation process would not be easily and perhaps impossible to distinguish. In fact with baseless collation there is no unique way to estimate these changes, since a reference text has to be set every time when visualizing the result.
   Therefore the tasks to be done after the collation are considerable. It might be better and easier to select iteratively a base text randomly, and do the collation instead of collating all the documents at a time.

After the collation it is up to the reader to determine a base text among the witnesses. If there are hundreds of witnesses having long text each one, it is obvious that determining a plausible base text would not be trivial. The visualization of the collation result would be confusing and tedious because of constant navigation between different witnesses. If two witnesses are displayed at a time, there will be need to have a reference witness along, but which one to select as the reference is also another issue that the collation process has to resolve. For instance if the displayed witnesses do not have words that are present in another witness, one would not instantly figure out the missing words, and would start wondering. Therefore a reference witness is mandatory for the visualization of baseless collation.
In fact the explicit and unique identification of the types of changes requires the presence of a base text against which witnesses will be compared. This implies that, with baseless collation there is no unique way to estimate these changes. And this is obviously one of the major disadvantages of baseless collation. The CollateX library would not be able to explicitly report these types of changes because it is a baseless text based collation software. So to improve it, there is need to modify or update the design of the software by providing an option of selecting between baseless and base text based collation.

b. Base text based collation

This approach of collation supposes that a base text is set, and one needs to determine closest document(s) to the base text. Therefore one of the documents needs to be chosen as a base text against which the remaining documents would be collated one at a time. This approach is easier and more intuitive for visualization. Types of changes can be easily computed and displayed.

4. Algorithms for Identifying Types of Changes, and Linking Source Documents to Collation Result

I. Algorithm for Identifying Types of Changes

As mentioned previously, explicit identification of types of changes requires setting a base text. As consequence the CollateX library would not be able to report the changes separately and uniquely. So to improve it, there is need to modify or update the design of the software by providing an option of selecting between baseless and base text based collation.
Since CollateX does not offer methods to explicitly retrieve the types of changes, I was left with the option of post-processing its collation result.
An algorithm post-processing the collation result in order to derive the types of changes that have been made to a base text is described below.

i. Get align tokens of base and witness texts
ii. An empty token in the base text indicates that the corresponding token in the witness text is an addition
iii. An empty token in the witness text indicates that the corresponding token in the base text has been deleted
iv. If two tokens are same, but do not have same offset in both base and witness texts, then this is a transposition (in
case of repeated tokens, the first token is considered as transposed one)
v. If two tokens are different but have same offset, then this is a mutation
   a. If part of the token in the witness text is similar to the one in the base text, then this is a partial mutation
   b. Else it is a full mutation

II. Algorithm for Linking XML Source Documents to Collation Result

i. Read the source file into a string
ii. Introduce single space between text node and opening and closing brackets of tags
iii. Delete empty spaces between attribute names and the equal sign, and replace all spaces with single space
iv. Create a vector by splitting the string based on empty space criteria
v. Select a text from the collation result (converted into a string) and get its offset
vi. Then create a vector by splitting the collation result from the offset backward based on empty space criteria
vii. Count number of occurrences of the text in the vector of the collation result
viii. Locate the text in the vector of the source file string by considering the number of occurrences

This algorithm is simple and valid for any kind of textual documents comparison having any format, as long as the non empty words of the documents are identical.
In fact a collation system does not modify collated documents' words, apart from adding empty spaces wherever needed for the alignment.

5. Working Principle of a Basic Collation System

A basic collation system is based on the Gothenburg Model, which defines any basic collation process as a chain of distinct steps in a workflow. The Gothenburg model comprises tokenization, alignment, and visualization units. The tokenization unit splits each input document into tokens and applies some normalization if needed, then sends the tokens to the alignment unit for aligning the tokens. The visualization unit displays the result of the alignment according to user defined output type.
According to this model each unit is dependent on the layer above it. In this sense it is possible to divide the units into optional and required units. For instance the tokenization unit might contain a normalization layer that could be considered as optional sub-unit.

Referring to distributed environment such as the Internet, it is possible to view each unit as a (micro) web service whose input is the output of the service above it. In this case the services could be classified as optional and required services as well.

Following is a diagram of the Gothenburg model as defined in [1] for a basic textual documents collation process.

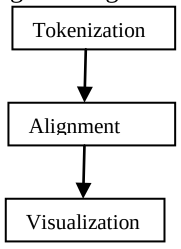


Fig. 1. Gothenburg Model for basic collation process

In this model the visualization unit/service is expected to offer users enough information in order to assist them in analyzing and interpreting the result. However the task of a visualization system is to format and display the data transmitted to it without carrying out any further computation or modification of the data. This model cannot distinguish between types of changes.

6. Extended Version of Gothenburg Model for Collation Process

The extended version that I propose is shown in fig.2. It consists of tokenization, alignment, interpretation, and visualization units.

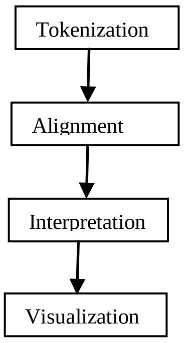


Fig.2 Document collation process steps

The interpretation of the collation result is so tightly dependent on the collation result that, it is practically not possible to effectively and robustly identify all types of changes that took place in the documents under collation, without having an interpretation module part of the collation process.
For this reason I have found it necessary to include the interpretation step within the basic collation process. Nonetheless it is still arguable about the required amount of interpretation that is needed to be part of the collation process. In this paper I have enumerated the four types of changes that textual document could undergo, but there might be more if we consider human's mind interpretations.
It is important to mention that a similar design for textual documents collation process based on Gothenburg model is described in [12].

7. Designing the System

Designing a software means figuring out how to implement functional and non-functional requirements that the software must perform. In short it is the process of building software that could represent something such as performing some tasks.
Based on usual documents comparison activities, the following requirements for user interfaces were set and implemented.

a. Possibility to collate documents as many as possible.
b. Possibility to collate documents of following formats: plain Text/XML/JSON.
c. Possibility of adding or removing any document from the collation at any time.
d. Possibility to set the output format of the collation result.
e. Possibility to attach precedence rules on document contents.
f. Possibility to use a style sheet to select text fragments from documents to be collated.
g. Possibility to open, edit and save any document from the collation.
h. Possibility to edit and save collation results (collation apparatus) locally or in TextGrid [6] repository.
i. Possibility to visualize the collation result by displaying base text and any witness side by side with differences highlighted.
j. Possibility to interactively alter collation result and re-collate.
k. Possibility to visualize and linked types of changes to respective collated texts.
l. Possibility to visualize the result in form of charts…etc.
m. Possibility to link the collation result to the collated source documents.
n. Possibility to provide help in decision making by providing statistics on the found changes.

8. Proposed Architecture for Interactive Collation Systems in Integrated Environments

The proposed architecture is a reflection of the design of a collation system using CollateX within the TextGrid project, as well as a standalone eclipse plugin that could be offered separately. For the standalone plugin particularly I made the identification of types of changes mandatory, because the goal of the plugin is not only to assist analysis of variations in textual documents but also providing computational support in decision making about the determination of a base text.
Because CollateX cannot accept XML as input, and for the time being there are no explicit methods for retrieving and distinguishing types of changes, I found that pre and post-processing steps were obviously unavoidable. In fact I met designers and developers of CollateX during the Interedition BootCamp workshop [7] in Darmstadt (Germany), and submitted my feature requests, which they promised to implement in the future.

However I think the identification and separation of types of changes (additions, deletions, transpositions, mutations,…etc) might require some changes or some adjustments of the currently implemented design. In fact CollateX is an implementation of the Gothenburg model for basic collation process. As argued in previous sections, a minimum statistical interpretation module must be part of the collation process in order to offer users sufficient information needed for analyzing and interpreting collation results.

For instance when I compare two hadiths (narrations) of Prophet Mohammad about same topic narrated by two of his companions, the critical apparatus should let me know what and how many differences are there, and where they are located. Based on this information I would be able to easily further my analysis and deduce a conclusion about the texts.

The following architecture (fig. 3) was designed and implemented.

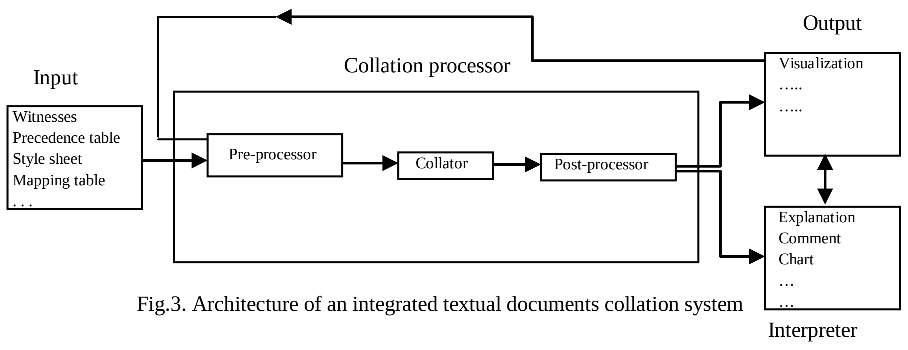


Fig.3. Architecture of an integrated textual documents collation system

Description

The system consists of three main blocks:

a. Input
It comprises documents (witnesses) to collate, plus some preprocessing files such as precedence table, style sheet, and mapping table for transforming and normalizing the witnesses before sending them to the collator. The interface is extensible if more pre-processing is desired.

b. Collation processor
It consists of three units:

   i. Pre-processing that mainly transforms and normalizes the input documents according to the collator instructions. It currently relies on purpose-based parsers.
   ii. Collator is the central processing unit of the system. It is presently based on the CollateX library which is able to align textual documents and produce a collation apparatus containing collation results.
   iii. Post-processor is responsible for transforming, formatting, storing, linking to source documents, and re-collating the collation results.

c. Output
It consists of visualizing and interpreting the result. There are two views available for visualizing collation results.
   i. Two text editors displayed side by side whereby the left editor contains the predefined base/reference text, and the right one holds one of the witnesses.
   ii. An editable browser that displays rendered HTML, and editable XML collation result.

The Interpretation unit comprises currently statistical analysis and their plots. It offers the following options:
a. Collation apparatus that enumerates and counts deletions, additions, transpositions, and mutations with their offsets with respect to the base text.
b. It further helps users locating positions of changed words in the displayed texts.
c. Moreover it assists users in effectively deciding about the proximity of documents. It can for instance inform users whether the collated documents have same origin, or they are completely different, or they are same but many alterations have taken place…etc. Degree of proximity is shown in terms of percentage. In other words the interpreter produces a report about the collation result.

Now that I have presented and explained the design and implementation of a collation system in an integrated environment such as TextgridLab, I would like to introduce a second architecture for textual documents collation process in a distributed environment.

9. Proposed Architecture for Interactive Collation Systems in Distributed Environments

The major difference between this architecture and the previous one is that, each unit is considered as a micro-service that might be located on its own server and geographically separated from other services.
One of the major problems of distributed systems is the interoperability between the existing services. How the services are going to communicate must be well defined and agreed upon. Usually a kind of data exchange format must be defined and exposed to all existing services. In this way interested clients could compose the services, and handle failures in order to obtain their desired results. However this method of calling the services one after the other requests every client to do it on its own, which might not be a trivial task for some clients.
In order to alleviate and simplify the clients' tasks, a broker between the services and the clients is needed. The service brokerage unit would take care of composing the services and handling failures by proposing or substituting some services with equivalent existing ones.

Specific constraints related to network availability and capacity, quality of available services, and specific types of data to be exchanged could be handled at the implementation level by the involved parties.

This architecture is an overall design of how the interoperability could be conducted. The implementation of this architecture is under construction.

The following architecture is an example of such a broker that mediates between users (clients) and the existing micro-services.

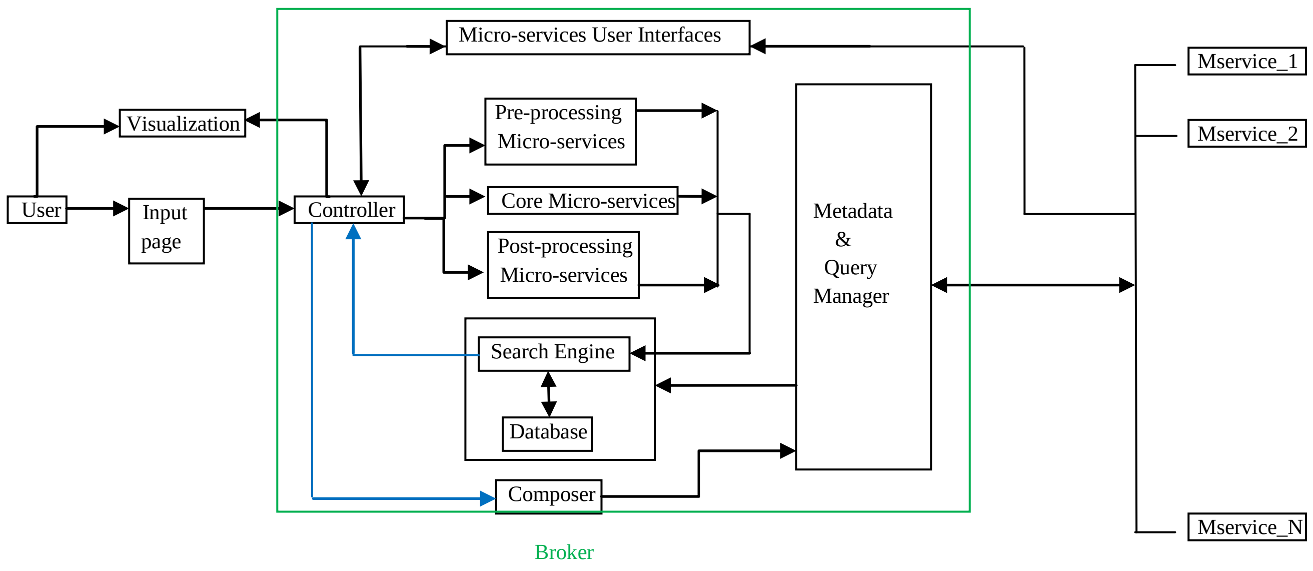


Fig.3. Proposed architecture of micro-services broker

Description

In the above diagram the TextGridLab could play the role of the Broker block, which would be an access point to all micro-services. In other words the proposed architecture will enable TextGridLab to act as a single access point to all textual documents collation micro-services. Through adequate user interfaces on micro-service basis, it will send queries to micro-services sites; collect query results and forward them to the visualization unit.

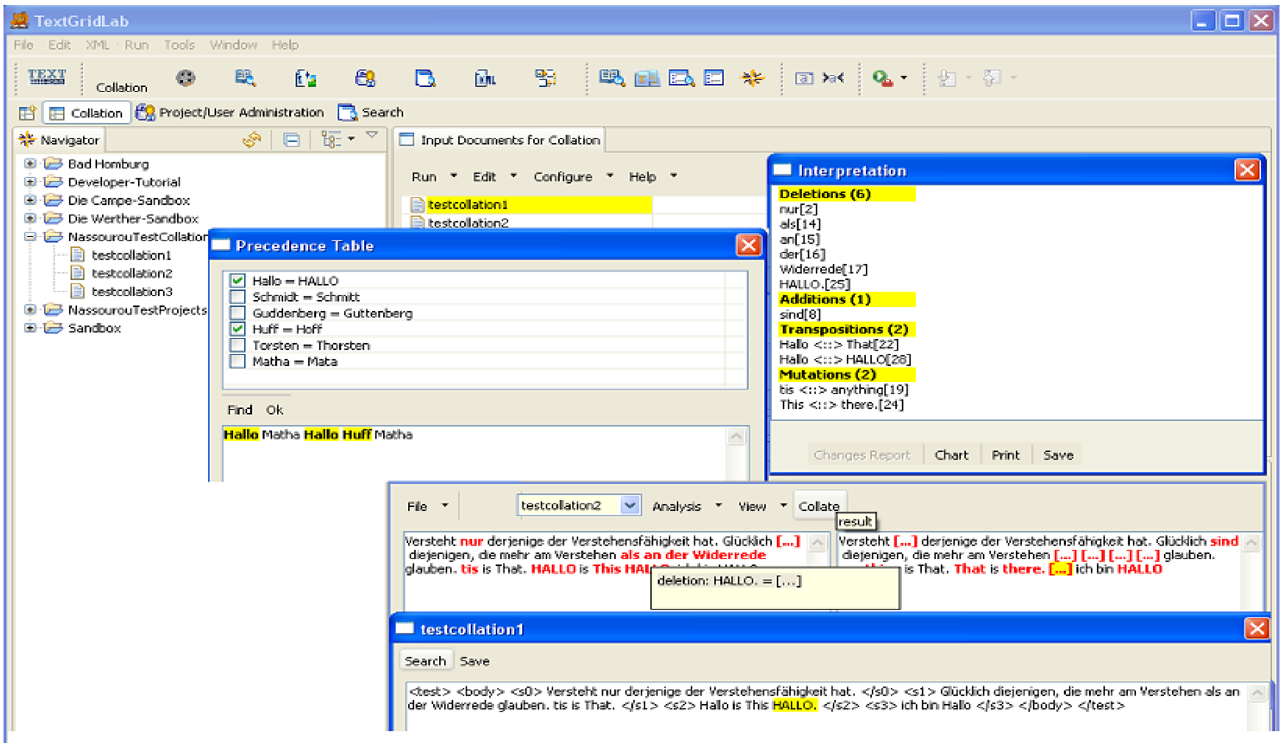
The **User** is a visitor of the TextGrid welcome page whereby all the micro-services are listed.
The **Input page** is the TextGrid welcome page.
The **Controller** unit is responsible for managing micro-services' user interfaces as well as identifying types of micro-services needed to accomplish the required job. Based on textual document collation process, I distinguish three types of services:

a. **Pre-processing Micro-services**
   These are services which are either required or optional for adjusting sent queries according to core services input format.
b. **Core Micro-services**
   These are services which are required for performing the desired task.
c. **Post-processing Micro-services**
   These are services which are either required or optional for visualization, analysis, and interactive collation.

The **Micro-services User Interfaces** module comprises web pages which are supplied by the micro-services and stored on the TextGrid server. Each project can update its user interface whenever needed.
The **Metadata & Query Manager** module is responsible for sending queries and collecting results from the micro-services. It also gathers all the relevant information about a micro-service, generates metadata for each one, and stores them in the **Database** unit.
The **Search Engine** collects search queries sent by the micro-service Controller, readjusts them if necessary, then queries the database to obtain micro-services qualities (for instance availability to process the sent queries), and finally sends back the queries with service qualities to the Controller.
After receiving queries with service qualities from the Controller, the **Composer** will normalize, format the queries according to each service input format, and then forward them to the Metadata & Query Manager for contacting the services.

10. Implementation

The programming language used to implement the interfaces was Java. Namely StyledText, Browser, Table, Combo box, and Menu widgets in SWT were employed for the realization of the architecture in integrated environment. Following is a sample view of the implemented interfaces.



- [✓] Hallo = HALLO
- [ ] Schmidt = Schmitt
- [ ] Guddenberg = Guttenberg
- [✓] Huff = Hoff
- [ ] Torsten = Thorsten
- [ ] Matha = Mata

Hallo Matha Hallo Huff Matha

Deletions (6)
nur[2]
als[14]
an[15]
der[16]
Widerrede[17]
HALLO.[25]
Additions (1)
sind[8]
Transpositions (2)
Hallo <::> That[22]
Hallo <::> HALLO[28]
Mutations (2)
tis <::> anything[19]
This <::> there.[24]

Versteht nur derjenige der Verstehensfähigkeit hat. Glücklich [...] diejenigen, die mehr am Verstehen als an der Widerrede glauben. tis is That. HALLO is This HAL

deletion: HALLO. = [...]

Versteht [...] derjenige der Verstehensfähigkeit hat. Glücklich sind diejenigen, die mehr am Verstehen [...] [...] [...] [...] glauben. is That. That is there. [...] ich bin HALLO

<test> <body> <s0> Versteht nur derjenige der Verstehensfähigkeit hat. </s0> <s1> Glücklich diejenigen, die mehr am Verstehen als an der Widerrede glauben. tis is That. </s1> <s2> Hallo is This HALLO. </s2> <s3> ich bin Hallo </s3> </body> </test>

Figure 4. sample interfaces of the interactive collation system

Figure 4 shows the TextGridLab user interface with the collation system interfaces. The precedence table window contains words with their substitutes during collation process, and a text editor for searching the substitutes in the collation documents' texts.

The interpretation window displays the collation apparatus by distinguishing and highlighting the types of changes with their offsets.

The bottom window shows a document called testcollation1 (in this case the base text) with the word "HALLO" highlighted to indicate that the word has been clicked in the collation result. The result is displayed in the window at the middle whereby the base text is shown on the left side, and the selected witness on the right side.

A tooltip window appears whenever a user focuses on a word with a red color. The window mentions that the word "HALLO" has been deleted in the witness as shown with empty brackets.

11. Conclusion and Future Work

This paper has presented a model for both integrated and distributed collation processes that improves the Gothenburg model. The model introduces an interpretation component for computing and distinguishing between the types of changes that documents could have undergone. Moreover two architectures implementing the model in order to solve the problem of interactive collation have been discussed as well. Each architecture used CollateX library, and provided on the one hand preprocessing functions for transforming input documents into CollateX input format, and on the other hand a post-processing module for enabling interactive collation.

Basic distinctions, pros and cons of baseless and base text based collations have been discussed. Simple algorithms for identifying types of changes, and linking source documents to collation result have also been introduced.

The next steps for completing this study are:

1. Completing the implementation of the architecture for the distributed environment.
   During the implementation issues related to network availability and capacity, quality of available services, and specific types of data to be exchanged would be handled.

2. Adding possibility to select between several collation engines in integrated environment.
3. Knowing that CollateX is an open source library, it might be advantageous to reuse the source code directly instead of JAR library. In fact in TextGridLab we use Vex editor source code as explained in [2, 3], so according to my opinion it might be better to reuse the CollateX source code in order to reduce the amount of post-processing steps. Of course time factor must be considered.

12. References

[1] http://www.interedition.eu/wiki/index.php/About_microservices
[2] Mohamadou Nassourou, "Understanding the Vex Rendering Engine", published on Eclipse Vex Wiki https://bugs.eclipse.org/bugs/attachment.cgi?id=178751, http://nbn-resolving.de/urn:nbn:de:bvb:20-opus-51333
[3] Mohamadou Nassourou, "Reference Architecture, Design of Cascading Style Sheets Processing Model" accepted in software engineering conference (SE 2011) by IASTED proceeding, http://nbn-resolving.de/urn:nbn:de:bvb:20-opus-51328
[4] http://wiki.tei-c.org/index.php/Textual_alterations
[5] www.cs.ucl.ac.uk/staff/g.piccinelli/eoows/documents/slides-zunino.pdf
[6] www.textgrid.de
[7] http://www.interedition.eu/wiki/index.php/Darmstadt_Bootcamp
[8] Theoretical Aspects of Textual Variations : http://www.bordalejo.net/DMU/ChapterIV.pdf
[9] Manfred Markus, "Getting to grips with chips and Early Middle English text variants: sampling Ancrene Riwle and Hali Meidenhad" , University of Innsbruck, ICAME Journal No. 23
[10] R. Mazza, "An Information Visualization Approach To Textual Variants", riccardo.mazza@lu.unisi.ch
[11] G. Silva, H. Love, "The Identification Of Text Variants By Computer", Inform. Stor. Retr. Vol. 5, pp. 89-108. Pergamon Press 1969
[12] http://wiki.tei-c.org/index.php/Textual_Variance#Analyzer