Abstract — Our implementation of a natural language processing framework (called TextProc) is described in this paper. We start with a general overview of the framework and continue with detailed description of its parts. Actual language processing is implemented as software plug-ins. Plug-ins can be put together into processes that perform a practical natural processing function. One such process is plagiarism detection, which is explained in detail. The process for plagiarism detection is actually used in digital library of University of Maribor and the integration of digital library with TextProc is also briefly described. At the end of this paper some ideas for future development are given.

Keywords — natural language processing, text processing, text mining, plagiarism detection, software framework, Slovenian language

I. INTRODUCTION

TextProc (abbreviation of “text processor”) is a natural language processing framework, developed as a part of master thesis at Laboratory for heterogeneous computer systems, Faculty of Electrical Engineering and Computer science of University of Maribor. The framework itself is language independent, since the framework doesn’t do the actual language processing by itself; natural language processing is done by software plug-ins.

There are two main ideas for its implementation. First, natural language processing is implemented in the form of software plug-in once and then reused many times. An expert in a specific natural language processing field implements a plug-in and other users can use it without knowledge of its implementation. Plug-ins are also developed in such a way, that it is possible to put them together into processes (we call them TextProc processes), which perform a more complex natural language processing operation. Second, the framework assures an easy way to build and execute before mentioned processes and also makes them available through several software interfaces for integration. This way, TextProc processes can be used in other applications, which make TextProc even more useful. Some ideas are based on a similar natural processing framework, called GATE (General Architecture for Text Engineering) [1][2].

Technically, TextProc is implemented in C#, based on Microsoft .NET Framework 1.1. The TextProc framework itself and some plug-ins also use Microsoft SQL Server 2000 database or better, although plug-ins can use any kind of database or other data storage method. Output for TextProc is mostly in XML format, so XSLT (XML transformation) together with web technologies like XHTML, CSS and JavaScript are used a lot.

The TextProc architecture is described in the second chapter. Third chapter presents the structure of a TextProc document. It is used for intermediate and final results. It is sent among plug-ins and defines the only way plug-ins communicate with each others. Plug-ins can be put together into processes that perform a practical natural processing function. One such process is plagiarism detection, which is explained in fourth chapter. Concluding remarks are done in the last chapter.

II. ARCHITECTURE

TextProc consists of several modules, as is shown on Fig. 1. Let’s start with TextProc plug-ins. TextProc knows five types of plug-ins, where only one type is actually meant to implement natural language processing algorithms. Other plug-in types perform reading and writing operations on various data sources, like databases, web pages and file systems.

Plug-ins must be implemented according to the software interface that is defined in second module, called ITextProc, where the letter “I” stands for “interface”. ITextProc also defines the TextProc document object, that contains the plain text and all other data about the document been processed, including all results that plug-ins produce. This document is then passed among the plug-ins and defines the only way plug-ins communicate with each other. The TextProc document is explained in details later. ITextProc module also includes all the functionality that has proven to be generally useful when implementing plug-ins, like precise timers, HTML code manipulation, simplified functions for hashing algorithms, XML transformations, reading and writing files and others.
Both ITextProc and plug-ins are implemented as .NET assemblies, also known as DLL files (Dynamic-Link Library). ITextProc must be used by plug-in developers for proper implementation. When a plug-in is developed and compiled, the resulting DLL is copied into a special folder of the TextProc framework. If the plug-in is properly implemented, the framework loads it and makes it available for use in TextProc processes.

The core module of TextProc framework is of course TextProcCore. It implements everything in TextProc framework that is not implemented in already mentioned modules or it is not a part of GUI (Graphical User Interface) directly. Its main purpose is to manage and run TextProc processes that are made of plug-ins. This module (together with ITextProc and plug-ins) can also be used directly for integration with 3rd party applications, if they are capable to load and run .NET assemblies. With just a few lines of code a developer can load and run a selected TextProc process and receive the result. The result can be in the form of the TextProc document object or as XML. Since XML can be transformed using XSLT, the end result that the developer receives can be almost in any form, making development of 3rd party applications as easy as possible. This presents the first possible method of integration.

Previously mentioned modules are enough to run TextProc processes, but to use TextProc directly by the user it requires additional user interfaces. TextProc provides two user interfaces: a desktop application and a web application.

TextProcGUI is a desktop application and is the main user interface for TextProc. Its primary purpose is to enable creation, management and execution of TextProc processes. It is intended for execution of long running processes on a large collection of documents. It also enables management of processes, available via TextProcDemo and TextProcWS (described later). The main window of the TextProcGUI is shown on Fig. 2. In the “Log” section (bottom) the event log is shown, where exceptions and other information is displayed. For instance, some plug-ins write their statistics at the end of processing in the log window.

In “Process” section a list is shown, where the first three items are always shown: process, corpus and script. If the first item is selected, then some basic settings of TextProc process are shown (on the right of the list). From here TextProc process can be started (with “Run” button) and the progress can be observed. There are two progress bars: for progress of the current document and for the entire corpus. If the “Corpus” list item is selected, we get a list of documents that are about to be processed. Documents can be loaded from the file system or from various databases; currently Microsoft SQL Server 20xx and MySql databases are supported. If the “Script” list item is selected, we get a multiline text input field, where a simple domain specific scripting language can be used. This scripting language is used for making changes to settings of used plug-in while the process is running, restarting the process and saving and resetting the event log. This way it is possible to automate experiments, where a single process restarts itself multiple times with prerecorded changes to plug-in settings. Such processes can then run for days without human intervention.

All other list items in the “Process” list are plug-ins. Selecting one of them shows all settings of the selected plug-in. Plug-ins can be added to the list and thus to the TextProc process by right clicking on the list and selecting a plug-in. A single plug-in can be added to the process multiple times and each instance can have different settings. Settings of a plug-in are actually all public properties of a C# class that implements the plug-in interface. For now only primitive data types are allowed.

TextProc is also available over the web in two parts. First
part is a web application called TextProcDemo that is primarily intended for presenting TextProc capabilities. It is also possible to use it for validation of correct implementation of plug-ins and processes. Some TextProc processes are published and publicly available to everyone for testing purposes. Web page for testing is shown on Fig. 3. TextProcDemo is currently available only in Slovenian language; Fig. 3 was translated for this paper.

TextProcDemo

Select one of the processes for text processing and enter some text in Slovenian language. After you click the "Run" button, the result will show in a new window.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>XML: Name</td>
</tr>
<tr>
<td>Script</td>
<td>reset</td>
</tr>
</tbody>
</table>

### Process 11: Same the text for plagiarism detection. Returns document ID (text).

**Input text:**

*Fig. 3: TextProcDemo*

First, the user selects one of the available TextProc processes. At the end of short description (in the dropdown list) the output format is mentioned and mostly the output is in XML format. XML can be transformed using XSLT, so the next dropdown list includes all available transformations for the selected process. Since TextProcDemo is intended for testing purposes, most transformations produce a human readable output, thus in XHTML format. The third input expects commands in the form of a script. Most processes don’t require a script, but some do. If the script is required for process execution, then those lines of script (with default values) are displayed when the process is selected, as is also shown on Fig. 3. This way the user knows that the script is required. The forth input is for the text, that is about to be processed. There is also an option to add data about used plug-ins into the resulting XML. Full name and version of used plug-ins is added, including execution times for each plug-in, which can be used for plug-in performance comparison. Once all the data is entered, we can press the “Run” button and the results are shown in a new browser window. This way the process settings and entered text remain entered and enable the user to make small changes to settings or text for another run. Since each result opens in a new window, it is also possible to compare results of different process executions.

The second part of TextProc that is accessible on the web is TextProcWS. This is a .NET web service that can be used to run exactly the same TextProc processes as TextProcDemo. Input parameters are also the same, where process and XSLT file are determined by the identification numbers. Users can get those numbers from dropdown lists on TextProcDemo web page. In case of Fig. 3, the process identifier is 11 and XSL file identifier is 1 (no XSL selected, result will be in XML format).

### III. TEXTPROC DOCUMENT

The idea of plug-ins in TextProc is that each plug-in performs some work, using the results of other plug-ins. This requires some kind of communication between the plug-ins. This is done through a special object we call TextProc document. It is a document structure, that carries all the results of the plug-ins. When a TextProc process is executed, the used plug-ins are executed in the given order, as is determined by the process creator (user). At the start, TextProc document is created, filled with plain text and passed to the first plug-in. A special kind of plug-in is responsible for getting the plain text and this plug-in is a part of the document object, not the process.

After the document object is filled with text, the first plug-in in the process is executed. It does its job and saves its results in the TextProc document, which is then passed to the next plug-in and so on to the end of the process. Each subsequent plug-in can use the results of the previous one. The last plug-in in the TextProc process is usually a special type of plug-in that is responsible for storing the results. In most cases results are saved as TextProc document in XML format. This can be written to a file or to memory for further processing, as is the case when using TextProcDemo or TextProcWS.

TextProc document contains following data:

- Reference to the plug-in that delivered the content in plain text. As already mentioned, there are multiple types of TextProc plug-in and one type is responsible for plaintext delivery.
- Various metadata. Plug-ins can add on change this data.
- Actual content as plain text.
- Sets of tags.

Sets of tags are what plug-ins use for actual data exchange. TextProc document can carry any number of sets that are identified by name. Each set can carry any number of tags. A tag is an object that points into the text and carries a set of values (strings). Tag object is defined in ITextProc module and contains the following data:

- Tag identifier (a number).
- Position of first character in the plain text that is tagged by this tag.
- Position of the last character in the plain text.
- Length of tagged text, calculated from positions of the
Advances in Communications, Computers, Systems, Circuits and Devices

first and last character.

• Line number of the first tagged character.
• A set of child and parent tags. A tag can point to other tags or be pointed to by other tags. This enables building hierarchies like: a word is a part of a sentence and a sentence is a part of a paragraph.
• A key – value collection of strings.

Since tags are in named sets, and values are in a key-value collection, those names and keys must be known in advance. Because of this, plug-in implementation somewhat depends on implementation of other plug-ins. This means that the order of plug-ins in a TextProc process is predetermined. For now, correct order of plug-in is not checked automatically and must be taken care of by the user, when the process is been assembled.

IV. PLAGIARISM DETECTION

There are many practical TextProc processes available for use. The TextProcDemo alone has 14 published processed. One of the most useful practical applications of TextProc for now is plagiarism detection. It is also the only TextProc process currently used by external software (described later). Plagiarism happens, when someone copies some content from other authors and then claims it as his own original work. Plagiarism is stealing and is therefore illegal. Plagiarism detection is a method of finding content that has been copied from others. A more detailed description is plagiarism, why it happens and the problem it causes is available in [4] and [5]. Plagiarism detection in TextProc is implemented using two TextProc processes. The first process transforms all documents into a form, more suitable for plagiarism detection. This transformation is as follows (each list item is a plug-in):

1. Text is tokenized, that is broken into words. This is a generic plug-in and breaks the content up regardless of its meaning. Because of this, certain content is broken, that from a human perspective should not be, but this problem is then solved by the second plug-in.
2. Some tokens are merged back together by a given set of rules. For instance, a decimal number “3.14” is separated by the first plug-in. The second plug-in determines that the dot is not a sentence separator, so it is merged into one word.
3. All words are converted into lemma form (canonical or dictionary form of the word). This is the most language specific plug-in in the whole process. Lemmatization is used because Slovenian language is heavily inflected; a word can have a very different form, depending on gender, case and number of the word.
4. Sentences and clauses are determined.
5. Paragraphs are determined.
6. Words in lemma form are merged into new sentences without redundant spaces, tabs or line reads that may be present in the original text; only a single space character is used as word delimiter. Also, words are sorted alphabetically on the level of a sentence. This way, word order within sentences becomes irrelevant.
7. Newly constructed sentences are hashed using a hash algorithm. Currently MD5 (Message-Digest algorithm 5) is used; several variants of SHA algorithm (Secure Hash Algorithm) are already supported.
8. Previous plug-in in called again; this time it hashes whole paragraphs.
9. Documents and its hash values for sentences and paragraphs are stored in a database.

This process is executed for each document separately. The second process contains only one plug-in and all it does is searches for hash values from one document that are also present in other documents of the same corpus. The end result is similarity report in XML format. Similarity is calculated as quotient between the length of similar content and length of entire document, expressed as percentage.

Plagiarism detection is currently used by the Digital library of University of Maribor (DKUM). TextProc has been integrated with DKUM using web service, available as part of TextProcWS. Each night (in times of least web traffic) DKUM sends unprocessed documents to TextProc and requests similarity reports in XML format. Those are then processed and saved in DKUM’s database for later use. Reports are saved in such a way that it enables progressive plagiarism detection. This means that a similarity report for a new document updates all reports of the older documents. Similarity reports are then shown on the administrative pages of DKUM, described in detail in [3].

Till now, plagiarism detection is done only between documents in DKUM corpus. In future, other sources will be added. This will probably bring the need to compare document between corpuses, which is now missing. Current implementation of plagiarism detection also lacks the ability to determine citations that are allowed and are thus not illegal. This is something we are still working on and is an essential feature for good plagiarism detection.

There are also other ideas for improvements. For instance we would like to detect numbers in text and replace them with a text or tag like “[number]”. For plagiarism detection, actual numeric values are irrelevant. If someone does plagiarism deliberately, then they probably make small changes in numbers, like adding a decimal, removing it, maybe changing it or writing it differently. Replacing the number with a tag makes such changes irrelevant and detectable. We already developed a plug-in that is able to tag all kind of numbers and other numeric values (dates), even those written as words (for example 1 written a “one”). All that we need now is to replace those numbers with a tag. This replacement would be done somewhere before plug-in number 6 (new sentence creation) in the first TextProc processes of plagiarism detection.

V. CONCLUSION

In this article, the existing capabilities of the TextProc natural language processing framework were presented. This
framework has already seen some practical usage, especially in the role of plagiarism detection system that is integrated into a digital library. There are ideas to implement a separate web application that would offer plagiarism detection service to everyone, again based on TextProc. Actually this is already possible by using TextProcDemo, but the current user interface in generic for all TextProc processes and is thus not effective enough for uploading large number of documents. A specific web application will be implemented, including registration, login, document management and report review functionality, targeted at average users (meaning non natural language processing experts). There are also other existing applications that we were already thinking to enhance them with TextProc. One of them is a 2.0 version of proprietary question answering system, currently in development (the first version is described in [6]).

TextProcDemo has also proven to be very handy for testing and presentation purposes, since it is easily accessible (over the internet; no software installation is required). It is also easy to use; users only need to select a process and enter some text. Although results are mostly in XML format, those can be transformed using XSLT into almost anything, including into good looking, human readable format and some of those are already available for use. With further plug-in development it is easy to extend the capabilities of TextProc and its use in the future.

REFERENCES


