

CHARLES UNIVERSITY IN PRAGUE
FACULTY OF MATHEMATICS AND PHYSICS

NPRG023 – Software Project

Project documentation

FilmTit

authors: Karel Bílek
Josef Čech
Joachim Daiber
Jindřich Libovický
Rudolf Rosa
Jan Václ

supervisor: Vladislav Kuboň

academic year: 2011/2012

Contents

I	Documentation	1
1	Introduction	5
2	General Introduction to the Area	7
2.1	Movie Subtitles	7
2.1.1	Subtitles in General	7
2.1.2	Translating Subtitles	8
2.1.3	Subtitle Formats	8
2.1.4	Current Tools	9
2.2	Translation Memories	10
2.2.1	What is a Translation Memory	10
2.2.2	Usual Implementation	12
2.2.3	Current TM Tools	12
3	Project Structure	15
3.1	General Architecture	15
3.2	Sharing the Implementation among the Project	17
3.3	Usage of Third Party Libraries	18
4	Building the Corpus	24
4.1	Retrieving Data	24
4.2	The Initial Data Properties	25
4.3	Loading Subtitles, Parsing	27
4.3.1	Loading Subtitles	27
4.3.2	GUI Parsing	28
4.3.3	Parsing	28
4.4	Sentence Splitting	29
4.5	Aligning the Subtitles	30
4.5.1	File to File Alignment	31
4.5.2	Subtitle to Subtitle Alignment	34
4.5.3	Approaches	34
4.5.4	Evaluation of Alignments	37
4.6	Running the Alignment	40

5	Core Translation Memory	41
5.1	Preliminaries	41
5.1.1	Choosing a DBMS	41
5.1.2	Retrieving Media Source Information	42
5.2	Architecture of the Core Translation Memory	42
5.2.1	Backoff Translation Memories	42
5.3	Candidate Retrieval	43
5.3.1	Translation Pair Searchers	43
5.3.2	Tokenization	43
5.3.3	Signature-Based Retrieval	45
5.3.4	Signature-Based Retrieval: Exact Matches	46
5.3.5	Signature-Based Retrieval: Named Entities	46
5.3.6	Full-Text Search	49
5.3.7	External Services	49
5.3.8	Statistical Machine-Translation Based on Moses	50
5.4	Candidate Ranking	50
5.4.1	Training the Models	51
5.4.2	Ranking for Exact Matches	51
5.4.3	Ranking for Full-Text Search Matches	51
5.5	Merging Similar Candidates	52
5.6	Technical Issues	52
5.6.1	Concurrency	52
5.6.2	Keeping the Database Up-To-Date	53
6	Training Machine Translation	54
6.1	Initial Setup	54
6.1.1	Initial Settings	54
6.1.2	Initial Results	55
6.2	Changes in the Setup	56
6.2.1	Tuning Set Size	56
6.2.2	Turning Off the Filtering	56
6.2.3	Pruning the Phrase Table	56
6.2.4	Binarizing the Models	57
6.2.5	Smaller Cube Pruning	58
6.2.6	Multithreading	58
6.2.7	Virtual Machine	59
7	User Space	61
7.1	Preliminaries	61
7.2	Architecture	61
7.3	Functionality Overview	62
7.4	User Management	63
7.5	Implementation Details	64
7.5.1	Data Types Overview	64

7.5.2	Database Mapping	65
7.5.3	Providing Feedback to the Translation Memory	65
7.5.4	OpenID	67
7.6	Exporting Subtitle Files	68
8	Communication between GUI and US	70
8.1	Remote Procedure Calls	70
8.1.1	RPC in general	70
8.1.2	Our implementation	70
8.1.3	Some issues	71
8.2	Manipulating Documents	72
8.2.1	Sending translation results – discussion	72
8.2.2	Whole Document Operations	74
8.2.3	Source Subtitles Operations	75
8.2.4	Target Subtitles Operations	75
8.2.5	Document Creation	76
8.3	User Registration and Login	76
8.3.1	Simple Login and Registration	77
8.3.2	Login via OpenID Services	78
8.4	User Settings	79
8.4.1	Account and Logging in Settings	79
8.4.2	Translation Workspace Settings	80
8.5	Remote Logging	80
8.6	Exceptions Thrown by RPCs	80
8.6.1	InvalidSessionIdException	81
8.6.2	InvalidDocumentIdException	81
8.6.3	InvalidChunkIdException	81
8.6.4	InvalidValueException	81
8.7	Chunk and Annotations	81
9	Graphical User Interface	85
9.1	Goals	85
9.2	Google Web Toolkit	85
9.2.1	Reasons for Using GWT and Discussion	86
9.2.2	Browser Support and Optimization	87
9.2.3	Designing by UiBinder	87
9.2.4	Twitter Bootstrap Library	88
9.3	GUI Structure	88
9.3.1	Gui.java	88
9.3.2	Pages	89
9.3.3	Dialogs	92
9.3.4	Remote Procedure Calls Implementation	92
9.3.5	Offline Mode	94
9.3.6	SubgestBox	96

9.3.7	Playing the Movie Files	97
9.3.8	Parsing and Segmentation	98
9.3.9	Manipulating with the Documents and Subtitle Items	98
9.4	Logging	99
9.5	Typical Usage	99
9.5.1	Document Creation	99
9.5.2	Document Translation	100

II Development Process Documentation 105

10 Implementation and Development Process 106

10.1	Timeline	106
10.1.1	Initial Project Meetings and Early Implementation Decisions	106
10.1.2	Early Development Process	107
10.1.3	Introducing the Shared Classes	110
10.1.4	The Main Development Phase	111
10.1.5	Final Development	116
10.2	Evaluating the Development Process	117
10.2.1	Technologies	117
10.2.2	Structure	117
10.2.3	Efficiency, communication	117
10.3	Work Distribution	118
10.4	Possible Future Development	119

III Manuals 120

11 Technical Manual 121

11.1	System Requirements and Setup	121
11.1.1	Running the Server	121
11.1.2	Database Configuration	121
11.2	Tasks	122
11.2.1	Step-by-Step Guide: Setting up the Server with an Existing Database Dump	122
11.2.2	Importing Data	124
11.3	Configuration	125
11.3.1	General Settings	125
11.3.2	Database	125
11.3.3	Text Processing Models	125
11.3.4	Data Import	126
11.3.5	Module-Specific Options	127
11.4	Adapting the TM to new Language Pairs	130
11.4.1	Basic Setup: Exact and Fuzzy Matching	130
11.4.2	Advanced Setup: Program-based Signatures and Machine Translation	131

11.5	Running the Moses Server	131
11.5.1	Installing the Moses Server	132
11.5.2	Running the Moses Server	132
12	User Manual	134
12.1	About the FilmTit Application	134
12.2	Installing Java and VLC Plugin	135
12.2.1	Installing Java	135
12.2.2	Installing VLC Plugin	135
12.3	Registration and Login	136
12.3.1	Registration and Basic Login	136
12.3.2	OpenID Login	138
12.3.3	Forgotten Password	138
12.4	Changing the User's Settings	139
12.4.1	Account and Logging in	139
12.4.2	Translation Workspace	140
12.5	Creating a New Document	141
12.6	Document Editing	142
12.7	Offline Mode	143
12.8	Operations with Documents	145

Part I

Documentation

Glossary

In this section, our terminology is described.

User

Everyone, who logs into the application, is a *user*. Each user has their own settings, authentication data etc., and can own multiple subtitle files, called *Documents*.

Document

A *document* corresponds to a subtitle file, owned by a user. (This is not connected to files that are used for building the corpus initially.) The document contains information about the *media source* of the subtitle file and a list of the *chunks* of the subtitle file which either are waiting for translation or have already been translated – in such case, it also contains their *user translations*.

Media source

A *media source* denotes a movie or a TV show, which is the source of a document. It contains the name of the movie, its release year and a set of tags describing the genre of the movie.

Subtitle item

A *subtitle item* is a piece of text that has a time declaration in the subtitle file and is to be displayed at the given time when playing the movie.

Chunk

A *chunk* is a piece of text that is to be translated at once. As we will describe later, each subtitle item is split into one or more chunks.

Unfortunately, in the code itself, we also often use the term *chunk* for the whole subtitle item. This is an error on our part, caused by not exactly defining all the terms beforehand and just using the term *subtitle chunk* for everything.

Surface form

Every chunk has a *surface form* – that means the text that is being translated. The surface form is without any non-textual information – we store these in annotations.

Annotations

Every chunk also has *annotations* – non-textual information that might be used at some point, but is not sent to the translation memory for querying. They mark positions of named entities, original position of newlines and dialog marks (“-”) – none of these are thought to be relevant for translation.

Timed chunk

A *timed chunk* is a chunk that also carries timing information from its subtitle item and information about order in which it appears in the subtitle item.

Or, from the other point of view: a subtitle item is split into one or more timed chunks, which all have the same time declaration, but they are assigned 1-based indexes that denote their original order.

Translation suggestion

A *translation suggestion* is a piece of text that might be a translation of a given chunk, coming either from the Translation Memory or from Machine Translation. It typically also contains the matched text found in the Translation Memory, and a score.

User translation

A *user translation* is a piece of text produced by the user as the translation of a chunk. It may or may not be identical or similar to a translation suggestion.

Translation result

A document that is being worked on consists of *translation results*. Each translation result contains a timed chunk from the original subtitle file, and may also contain a list of translation suggestions or a user translation.

Page

A *page* in GUI (also often called “screen” by other authors) is a viewpoint in the GUI, having a distinct name, URL identifier, layout and function – such as the Help Page or the Translation Workspace.

Dialog

A *Dialog* is an element smaller than a page in height and width, displayed on top of a page and disabling the user interaction with the page. Similarly to a page, it also has a distinct layout and function. It typically contains text boxes to fill in or edit some values, and buttons to submit the values or to close the dialog.

Chapter 1

Introduction

Thanks to the effortless sharing of video files, there is a relatively big community of movie and TV series fans on the Internet who spend a significant amount of their free time translating the subtitles of foreign movies to their native languages.

There are many sophisticated commercial tools available that assist professional translators with their translation tasks, as well as software solutions used by big translation agencies; however, a complex tool for fan-based translation using state-of-the-art techniques is still missing.

In contrast to the area of commercial translation, where each translation agency keeps previously translated texts as proprietary knowledge, fan translations are based on a non-commercial community approach that creates huge amounts of translated subtitles that are publicly available. There has not been a tool for subtitle translation taking advantage of the public availability of this data.

The subtitles themselves are an ideal source of parallel data for linguistic research. They are used for the creation of parallel corpora for statistical machine translation (e.g. the CzEng corpus¹ used for training the statistical machine translation tools at ÚFAL). The film industry is also aware of this fact and there have been some attempts to do an automatic translation of subtitles with minimal post-editing based on existing subtitles data.²

Translation memories (discussed in more detail in Section 2.2) are in these days the most common tool used by translators. The memories are usually very domain specific to make the computation feasible on the translators' PC and to avoid any confusion in terminology. In the case of movie subtitles, there is no such terminological danger. Moreover, we believe that the similarity of the movie scripts will produce more data, leading to better results. These arguments make us believe that creating a translation memory based tool focusing on movie subtitle translation can be both useful and successful among the target users.

As indicated in the previous paragraphs, the goal of our project is to create an application that will help amateur translators of movie and TV show subtitles with their efforts. The core of the application is a large translation memory, which will be gradually extended and improved by users' translations. The translation memory exists only in one publicly available instance, which

¹<http://ufal.mff.cuni.cz/czeng/>

²Marian Flanagan (2009): *Using Example-Based Machine Translation to translate DVD Subtitles*. Proceedings of the 3rd Workshop on Example Based Machine Translation, p. 85–92

is on the server, and is therefore shared by all users. We focus on the English-Czech language pair; the application itself is language independent, but different training data would be necessary to run the application for another language pair.

Although our original goal was only to build a translation memory, we found that the collected data can be used in building a parallel corpus for training a statistical machine translation system. We built such a system using the Moses machine translation engine and added it into our project as another option for translators; we describe the system in Chapter 6.

The collected data can be also used for research on the language used in the subtitles in general.

The following chapter contains a general introduction to the area of producing subtitles and of computer-assisted human translation. The next chapter provides general information about implementing the project, including usage of external libraries. The following chapters contain detailed description of the individual project modules.

The second part of the documentation is devoted to the evaluation of the development process of the project, including the reasons for the decisions we took, and an overview of the work distribution among the members of the project.

The last part of the documentation contains the manuals. Chapter 11 contains a manual for a server administrator, explaining how to install the server part of the application. Chapter 12 contains the manual for web application users (which is also incorporated into the application as a Help page).

We would like to thank doc. RNDr. Vladislav Kuboň, Ph.D. for useful and practical advices during supervising the FilmTit project, Mgr. Milan Fučík for technical support of our virtual machine, RNDr. Ondřej Bojar, Ph.D. for advice with Moses, the admin of opensubtitles.org for the data, the testers for finding many interesting bugs, and KFC and Burger King for free wifi connection.

Chapter 2

General Introduction to the Area

2.1 Movie Subtitles

2.1.1 Subtitles in General

Subtitles are textual versions of the dialog in films or television shows which are shown on the screen at the same time as the dialogs are performed. The most common reason to show the subtitles is either to allow deaf or hard-of-hearing viewers to follow the dialogs or to allow viewers who are not speakers of the language used in the movie to understand the movie while hearing the original sound track. The major difference between these two types of subtitling is that the subtitles for deaf people usually contain not only the dialogs themselves, but also additional information, e.g. that music is playing or a description of background noises important for the story.

Originally, the subtitles used to be chemically corroded or laser burnt into the film tape. With the advent of digital technology, the subtitles can be added dynamically to the screen and do not have to be an actual part of the movie. Moreover, they can be distributed independently of the movie.

Subtitles on DVDs are stored in a bitmap format within the movie itself.¹ With the occurrence of unofficial movie file sharing, bitmap files became too big to distribute, so several unofficial formats, based on text, were created.

In those cases, the subtitles – now in text format instead of bitmap, as in DVD – are in a standalone file which contains the actual subtitles and some meta data telling the media players which subtitle items should be displayed at which time interval of the video playback.

We do not deal with the type of subtitle that are in non-textual formats at all; all subtitles we have in our database and that users translate are only textual.

¹Although official DVD documentation is not available online for free, you can find some information on the DVD project on sourceforge - http://dvd.sourceforge.net/spu_notes

<pre> 4 00:02:04,718 -->00:02:08,054 I just want to be alone with her and hold her and kiss her 5 00:02:08,179 -->00:02:12,309 and tell her how much I love her and take care of her. </pre>	<pre> {1025}{1110}I just want to be alone with her and hold her and kiss her {1375}{1460}and tell her how much I love her and take care of her. </pre>
---	---

Table 2.1: An example of the subtitle file formats, *srt* at the left, *sub* at the right side.

2.1.2 Translating Subtitles

Professional translation of movie subtitles and unofficial fan translation is very different for several reasons.

While making a professional translation of the subtitles, the translators can see the whole screenplay of the movie, accompanied with a lot of notes concerning the expressiveness of the utterances and using the rhetorical devices (culture references, sarcasm etc.), on the other hand they often do not see the movie at all.² They usually make changes to the timings to better fit on the screen, have the same number of syllables (in cases of dubbing), and so on. They are equipped with the same tools as other professional translators, such as translation memories or glossaries for checking the consistence of the used vocabulary.

On the other hand, fan translators use simple editors without any special functionality but they usually watch the movie during translation. They usually have pre-existing English subtitles from unofficial sources, which they simply translate without much retiming, subtitle splitting or anything else.

We want to target specifically the second kind of user.

2.1.3 Subtitle Formats

There are two formats of subtitles that are widely used in unofficial movie sharing.

The most widely used is the SubRip SRT format. Sometimes also MicroDVD sub format is used. An example of the formats can be found in table 2.1. According to the database we received from OpenSubtitles.org (the database will be described later), more than 99% of subtitles are in SRT format.

SRT or *SubRip* is a format from the eponymous GPL program, used mainly for converting the bitmap-based subtitles on DVDs to text data. There is no formal specification of the subtitle file format, perhaps because of its simplicity.³

²For an illustration, how script, given to a translator may look like, see <http://www.flickr.com/photos/fuxoft/5731288526/in/photostream/>; also you can read some articles about creating movie translations (in Czech) <http://www.fffilm.name/search/label/titulky>

³Some kind of “specification” can be found on the website of the Matroska codec, which can read *srt* files –

A *SRT* file does not have any header and contains everything as plain text. Each subtitle item consists of three or four lines and is separated from the previous and the next one by an empty line. The first line is the order of the subtitle in the movie, the second line contains the absolute time in the movie when the subtitle shows up and when it disappears. The time is expressed with the precision of milliseconds. The next several lines contain the actual text of the subtitle. It can also contain simple HTML-like formatting tags, although support for those varies across media players.

MicroDVD *sub* format is a format, used by the free but proprietary MicroDVD media player, which discontinued development in 2001. This format also lacks any formal specification and support is usually worse among players.

In the MicroDVD *sub* format, each line represents a subtitle item. Numbers of frames in the video between which the subtitle is displayed are used instead of the absolute time in seconds. After the frame numbers in brackets, the actual text of the subtitle follows. The pipe symbol is used as a line separator.

There are also some other proprietary formats having the extension *.sub*, but they look differently.

For our purposes, we can take note of one important thing, partly visible in the example on top: not every subtitle item is a sentence – sometimes, there are more sentences in one item, sometimes, there is one sentence split across many items. We describe how we dealt with this problem in the section 4.4.

In the examples in Table 2.1, a mixup of letters “l” and “I” can be noticed. This is indeed not an error on our part, but an error, caused by copying DVD subtitle to text file using programs like SubRip that use some light form of OCR to get the text information; and “l” and “I” often look indistinguishable in sans-serif fonts used often in the DVD subtitles.

Before the advent of fast internet connections, movies were often unofficially shared using just physical media; because of that, movies were sometimes split into multiple video files to be put on a separate CDs. For this reason, some subtitles are sometimes split in a half.

2.1.4 Current Tools

There are several freeware tools for amateur subtitle translation. They are usually standalone applications. The GUI of such applications typically consists of two columns, the first one with the subtitle items in the original language, the second one to be completed with the translation. Some of them offer also a synchronized playback of video during the translation. We did not find an application which would be web based and specialized on subtitles translation. An overview of such tools is in Table 2.2.

From our view there is also an interesting project by Google – *Google Translator Toolkit*. It is a web application designed to help the translators with their work which also supports parsing subtitles. During their work, a translator can see the document both in source and target language and is given per sentence suggestions from Google Translate or their own translation memory which the user has previously imported. After the work is finished, the created translations can be exported as a translation memory archive. This tool did not become popular among

<http://matroska.org/technical/specs/subtitles/srt.html>.

name	licence	platform	website
Subtitle Workshop 4	freeware	Windows	www.urusoft.net
Subtitle Processor	GPL	Windows	sourceforge.net/projects/subtitleproc
Open Subtitle Translator	GPL	Windows / Linux	sourceforge.net/projects/opensub
Gnome Subtitles	GPL	Linux	sourceforge.net/projects/gnome-subtitles
Subtitles Translator	freeware	Windows	www.mironto.sk

Table 2.2: An overview of subtitle translation tools available on the Internet.

translators, probably because of the quality of the machine translation provided. Professional translators would probably not be willing to bring their valuable translation memories to Google for free.

2.2 Translation Memories

2.2.1 What is a Translation Memory

A *translation memory* is a tool for computer-assisted human translation. The core idea behind translation memories is the assumption that sentences that are similar in the source language will likely have a similar translation in the target language. Then, if the tool is able to provide the translator with a translation of a similar sentence, there is a big chance that the provided sentence will require only small changes.

Translation memories are widely used in the translation industry, mostly while translating technical documentation and during the localization of software. In such cases the translators usually take advantage of the fact that the new version of the manual of a given software does not differ too much from the previous version. A survey⁴ among companies producing multilingual documentation for their products in 2006 showed that 82.5 % out of 874 such companies use a TM system.

While using a translation memory there is usually a tendency to keep the database as clean as possible in terms of domain – to contain only sentences relevant for translated topics. The reasons to do so are the effort to keep the database as small as possible in order to not make the database search too slow and not spoil the terminology that is used in the particular area. To keep the terminology consistent, domain glossaries are usually used.

There are several reasons why using translation memories makes the translators' work more efficient. The main advantage is that it reduces the cost and makes the translation process faster because the amount of the translator's work is smaller – the work that has been done once can be easily reused. It helps to keep consistency of translation between multiple documents and also consistency with the previous versions of documents (e.g. technical manuals). It is also quite easy to ensure that each sentence of the original document was translated into a segment of the target language document.

⁴Elina Lagoudaki (2006), *"Translation Memory systems: Enlightening users' perspective. Key finding of the TM Survey 2006 carried out during July and August 2006.* Imperial College London, Translation Memories Survey 2006, p.16

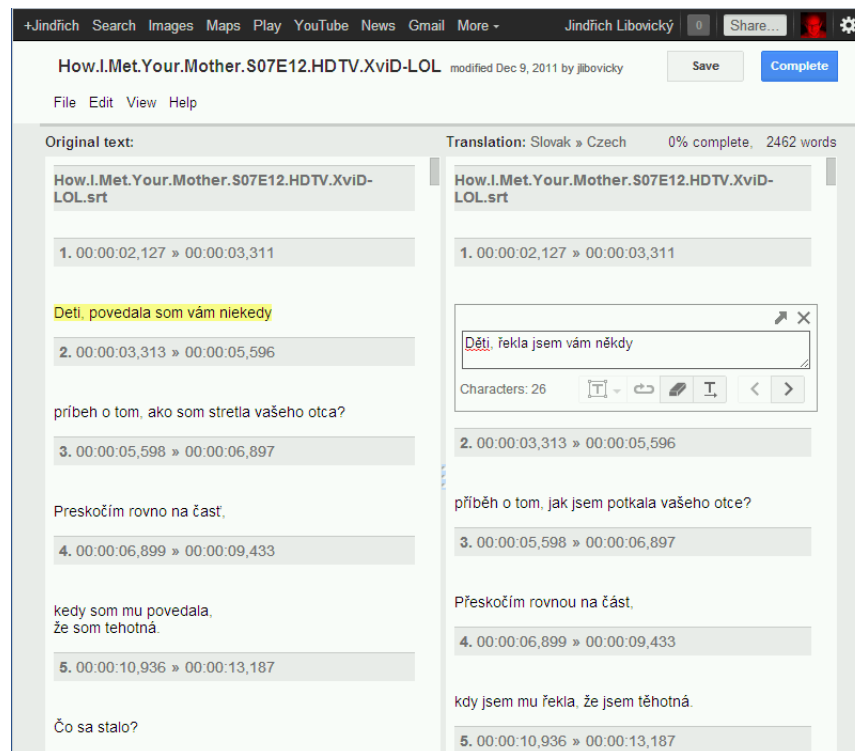


Figure 2.1: Google Translator Toolkit

There are also some obstacles in using translation memory systems. Professional Translation Memory Management Systems are very expensive and the maintenance of such system can be demanding as well. From the point of view of the quality of the translation, there is a danger that the translator could translate the text mechanically sentence by sentence instead of focusing on translating the message of the text.

In contrast to complete machine translation, it is still the human translator who controls the whole process of translation. Nevertheless, improvements in machine translation allow to provide a machine translation output together with TM candidates.

2.2.2 Usual Implementation

A simple option is to provide only candidate sentences where the sentences in the source language match each other exactly. Usually, the database is relatively sparse and probably no sentences would be retrieved. In such cases a fuzzy matching algorithm is used to retrieve also similar sentences where the similarity is usually a metric based on the Levenshtein editing distance.

Before using the translation memory itself, some preprocessing may be necessary. Often, text extraction is needed (e.g. in the case of localization of user interfaces). Sometimes, finding terminology or other named entities and segmentation to elementary units, usually sentences, is always done. This components can be either rule based or statistical.

During the translation process, the system retrieves similar sentences from the database of already translated sentences. Usually, sentences having the smallest letter based or word based *Levenshtein editing distance* are used. Despite this, we finally decided to use a different way of retrieving matches, this algorithm will be shortly described as the state of art in translation memories.

The Levenshtein distance of two strings is the minimum number of edits (insertions, deletions and substitutions) which is needed to transform one string to another. A modification called Damerau–Levenshtein distance allowing also transposition of two adjacent letters can be also used.

Originally a bottom-up dynamic programming algorithm was used – a distance of two strings is computed from the knowledge of the of the distance of one letter shorter prefixes. Later the Bitap algorithm used in Unix tool *agrep* appeared. Theoretically also a finite state machine (Levenshtein transducer) can be constructed for Levenshtein distance (see Figure 2.2).

For bigger databases the online algorithms begin to be very slow a preprocessing of the database is necessary. Some sophisticated indexing methods are used, among them the suffix trees or n -gram indexes. The candidates retrieved from such indexes are later examined more carefully.

2.2.3 Current TM Tools

The tool most frequently used by professional translators is *SDL Trados*. It was originally developed by the German company Trados which was later acquired by its biggest rival, British SDL. It became popular mostly because of its integration to Microsoft Office starting in the mid nineties.

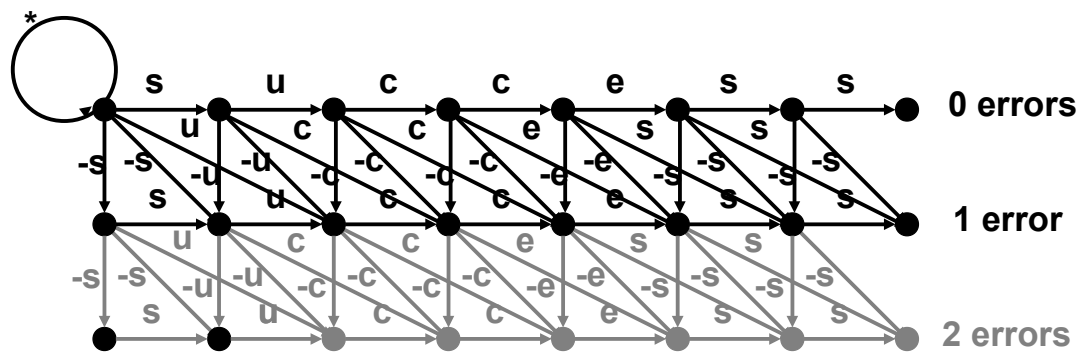


Figure 2.2: An example of the Levenshtein transducer for the word "success". (Taken from the study material for the *Information Retrieval Systems* course at MFF UK by Michal Kopecký.)

It is a very complex software. Except the actual tool for the end users, it includes central servers where a translation agency can gather its data and from where the data can be distributed to the translators' PCs and sophisticated tools for team work. Other popular translation memories are DeJaVu by the French company Atril and Wordfast by an American company of the same name. According to a 2006 survey⁵ 54 % of professional translators world wide used products by Trados or SDL, 17 % Wordfast and 16 % DeJa Vu.

There are also open source translation memories projects, mostly not very elaborate and with a small community. IBM has recently released its internally used software under the Eclipse Public Licence. A widely used open source tool is called *OmegaT*. It is a cross-platform software which provides very complex functionality. It can be interconnected with Trados servers, supports most of the translation memory formats and supports also many textual formats, but without integration to other software. It is also commonly used for translation of open source software. According to previously mentioned survey it is used by only 3 % of professional translators.

There is also a translation memory project which is somehow similar to our project. It is called *MyMemory* and it was created by the Italian translation agency *Translated.net*. It is a big general-purpose translation memory available as a remote service via the Internet. It contains translations from various domains, which can be used either separately or as a whole. The usage of MyMemory is free of charge, under the condition that the translations produced using the service will be provided as data to the service.

⁵Elina Lagoudaki (2006): *Translation Memories Survey 2006: Users' perceptions around TM use*. Translating and the Computer 2006, London: Aslib.

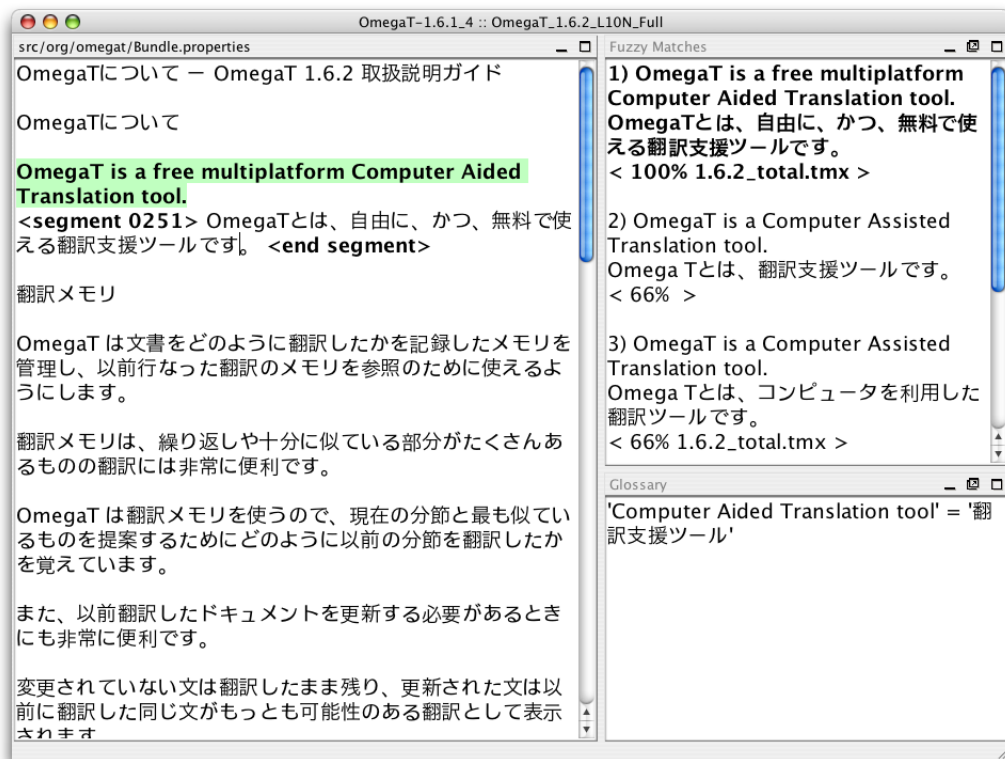


Figure 2.3: A screenshot of the OmegaT version 1.6

Chapter 3

Project Structure

3.1 General Architecture

The structure of the project can be best described as an implementation of a multi-tier architecture consisting of the Translation Memory Core, the User Space and the web application GUI, as can be seen in Figure 3.1.

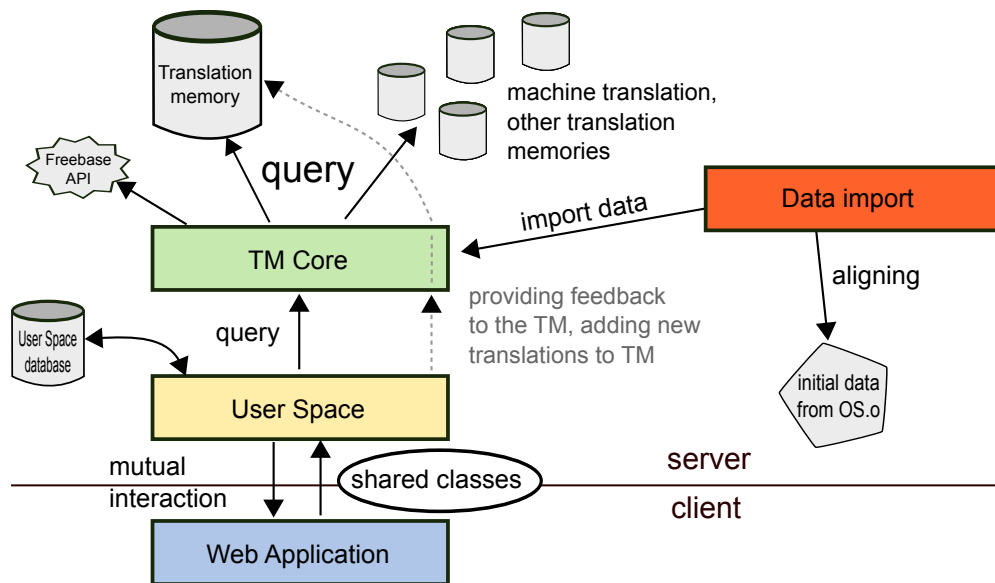


Figure 3.1: Scheme of the application architecture.

TM Core

The deepest level of the application is the *Translation Memory Core*, which operates directly with the parallel chunks stored in the database. It is implemented in Scala. It provides an interface for retrieving translation suggestions from multiple sources – different ways of processing our

own translation memory and the machine translation output. It also assigns a score for each suggestion based on the retrieval parameters (e.g. edit distance for exact retrieval or term vector similarity in fuzzy retrieval) and movie characteristics taken from the *Freebase.com* knowledge base.

The core layer also queries our machine translation server, which is running as a separate process and is using a phrase-based Moses translation engine.

Data Import

Data import is a separate module, for cleaning up and parsing the data from the OpenSubtitles.org database, aligning language pairs and using the Core layer to import the pairs into the database. The classes should be run only once to have the data ready; however, we found it useful to have those data-importing classes as a part of the whole application, rather than some external scripts because it allowed us to repeatedly change the data themselves and keep the code compact. This part is also implemented in Scala.

User Space

The middle-ware layer is called the User Space. Its task is to interact with the translation memory itself and mirror all GUI operations on the server side. The User Space is implemented in Java. It uses the same database as the Core and one database table is shared among them. The TM Core is used only as a service which is queried for translation suggestions. The interaction with the GUI is much more complicated because each operation from GUI has to be reflected in the US. The US provides a permanent storage of users' work to make the whole application including the users data available from the Internet. Except this function, the US provides the TM suggestion for the GUI.

GUI

The GUI module is written in Java, using *Google Web Toolkit*, which translates Java code into JavaScript and provides a framework for the simple implementation of remote procedure calls (RPCs) via the HTTP protocol, using POST requests. The server side of the GUI layer displays the appropriate JavaScript and CSS code to user's browser, which then communicates with the User Space through JavaScript AJAX calls through RPC, as is described above. The GUI lets the user log in, upload the subtitles, parses the subtitles into individual chunks, offers the user translations for every chunk and optionally plays the video of the chunks being translated.

Shared classes

GUI and User Space are both written in Java (even when GUI is then translated to JavaScript) and can, therefore, have some Java *shared classes*, which are used for communicating from the GUI to the User Space and from the User Space to the Translation Memory Core.

Other information

Translation Memory Core and the User Space are both parts of the same `.jar` file and are, therefore, run in the same Java process. User Space is running as a Java Servlet.

The server side of the GUI, which returns the HTML, CSS and JavaScript, is also in the same `.jar` file, together with the required GWT assets, and is run as a second Servlet. There is also a third servlet which is used for downloading the exported subtitles.

All Servlets are loaded using a *Jetty* web server. Maven is used for building the application and retrieving dependencies, the Jenkins tool was used for continuous building and testing of the project.

As noted above, a Moses machine translation server is running as a completely separate process.

3.2 Sharing the Implementation among the Project

Because all parts of the projects are Java based, we can share a set of classes between the GUI and the server to avoid unnecessary redundancy and to make the project structure clearer.

However, as noted in the GWT documentation,¹ not all Java classes are directly translatable to GWT. The main issues we encountered were:

- The shared classes cannot reference any class that is not translatable to JavaScript through GWT (for example, any third-party libraries).
- The shared classes can use only a subset of the Java Runtime Library. E.g. we could not use `java.util.regex`, which is not implemented in GWT; we had to use `com.google.gwt.regex.shared` instead.
- Serialization works differently in GWT. What this effectively meant for us is that, even when we use the general `List` interface for a property, we cannot set the value to an instance of a subtype of `List` that is not implemented in GWT. More specifically – we wanted to use `scala.collection.JavaConverters.AsJava[List]` for Scala \Leftrightarrow Java conversion, but it was not possible, since it uses its own implementation of Java `List` interface.

Therefore, shared classes could only use a subset of Java. Generally, what is translatable with GWT to JavaScript is also translatable by the Java compiler to JVM bytecode (except for direct insertions of JavaScript code of course), but not the other way around.

Structure of the shared classes follows the terms mentioned in the Glossary as can be seen in Figure 3.2. Detailed UML diagram of the shared classes is in Figure 3.3 and Figure 3.4.

¹<https://developers.google.com/web-toolkit/doc/latest/DevGuideCodingBasicsCompatibility> and <https://developers.google.com/web-toolkit/doc/latest/RefJreEmulation>

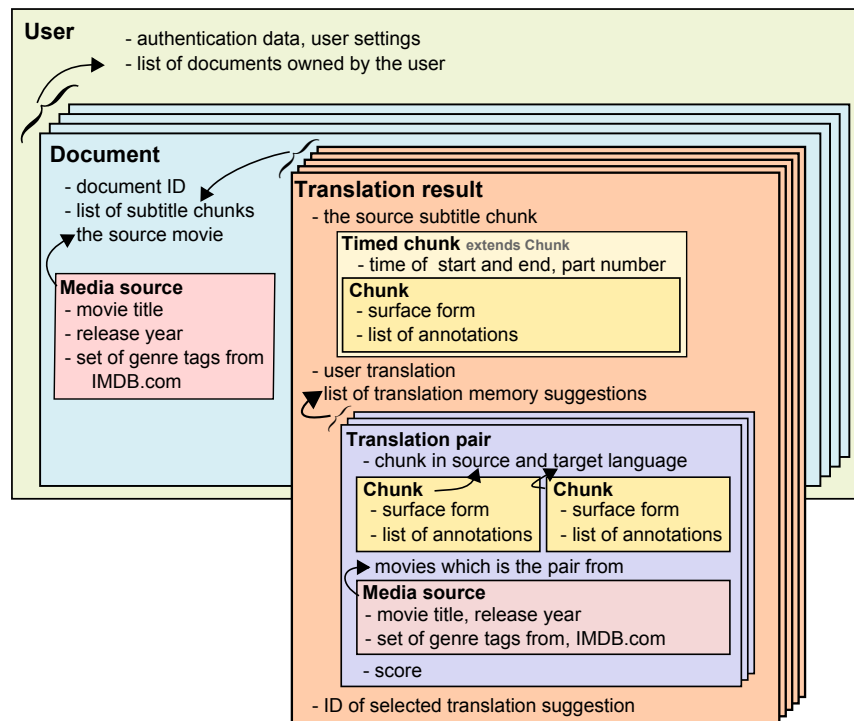


Figure 3.2: Scheme of the shared classes.

3.3 Usage of Third Party Libraries

We use a number of third party libraries and tools. Most of them are linked to the project as Maven dependencies. An overview of the used dependencies including their licenses and a short descriptions is provided below.

- **Scala Compiler 2.9.1** – Scala License
Compiler of the Scala programming language.
- **Posgres SQL 9.1.** – PostgreSQL License (similar to MIT and BSD licences)
The JDBC driver allowing the application to connect to a PostgreSQL database.
- **Google Web Toolkit 2.4.0** – Apache License 2.0
Google Web Toolkit is an open source set of tools that allows web developers to create and maintain complex JavaScript front-end applications in Java.
- **Hibernate Validator 4.1.0.Final** – Apache License 2.0
Hibernate Validator is the reference implementation for JSR 303 - Bean Validation. Bean Validation defines a metadata model and API for JavaBean validation.
- **JUnit 4.10** – Common Public License, v 1.0
JUnit is a unit testing framework for the Java programming language.

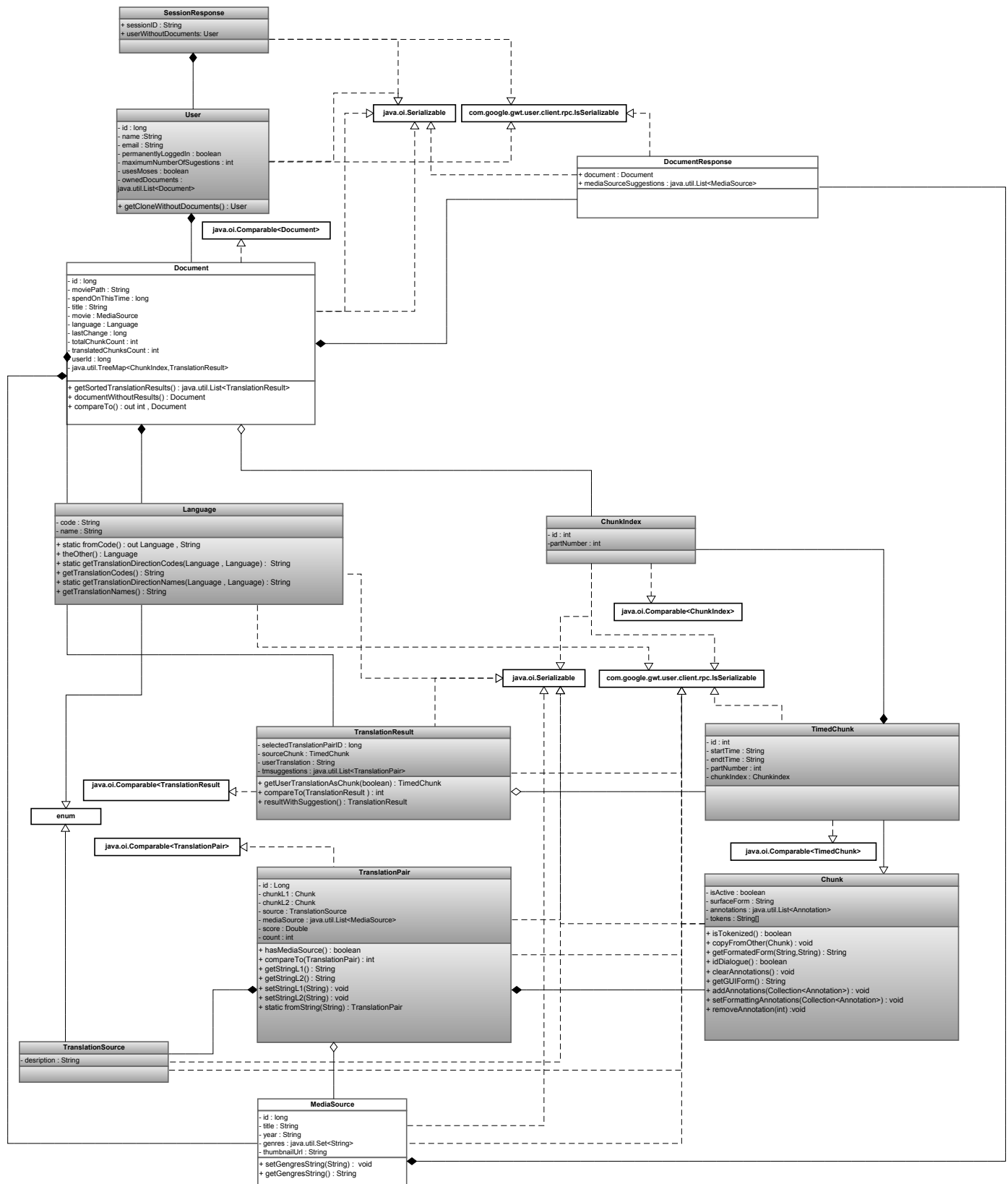
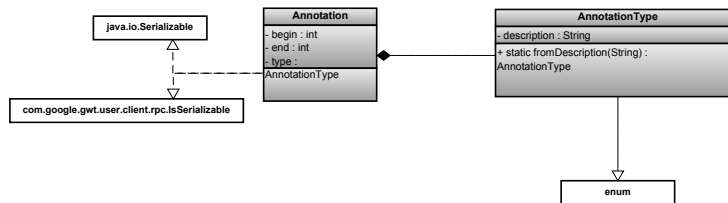
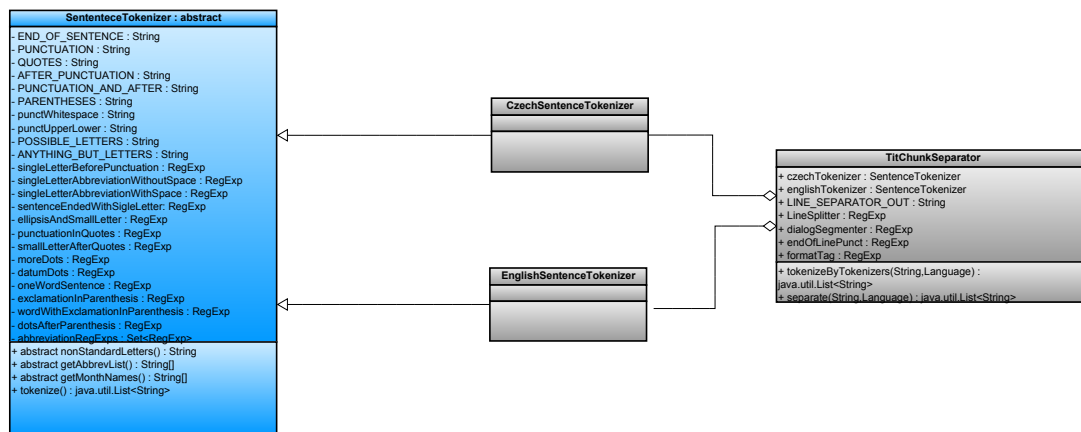


Figure 3.3: UML diagram of shared classes

Annotations



Tokenizers



Parsers

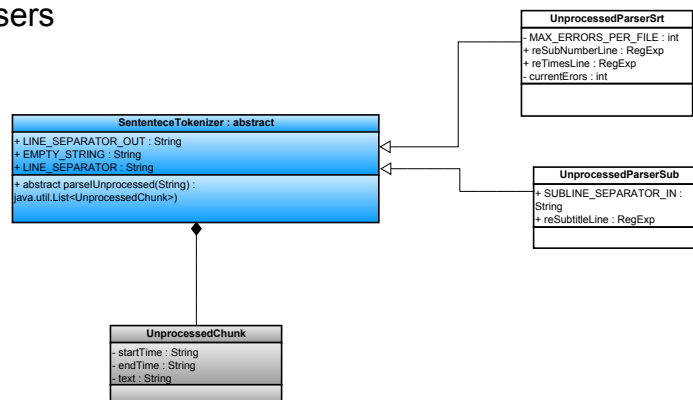


Figure 3.4: UML diagram of helping shared classes providing functionality needed among different modules

- **ScalaTest 1.6.1** – Apache License 2.0
ScalaTest is a unit testing framework for both Scala and Java programming languages.
- **language-detection** – Apache License 2.0
This is a language detection library implemented in plain Java developed by the Cybozu company.
- **Apache XML-RPC client** – Apache License 2.0
An implementation of XML-RPC, a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism.
- **Apache Commons** – Apache License 2.0
The Apache Commons is a project of the Apache Software Foundation, formerly under the Jakarta Project. The purpose of the Commons is to provide reusable, open source Java software. We use particularly the Lang3, Math and Validator components.
- **OpenNLP 1.5.2** – Apache License 2.0 or LGPL
The Apache OpenNLP library is a machine learning based toolkit for the processing of natural language text. It supports the most common NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and coreference resolution. These tasks are usually required to build more advanced text processing services.
- **HSQldb 2.2.8** – BSD License
HSQldb (Hyper Structured Query Language Database) is a relational database management system written in Java. It can be used just in the memory. (Used in tests.)
- **Google Guava 11.0.2** – Apache License 2.0
The Guava contains several of Google's core libraries: collections, caching, primitives support, concurrency libraries, common annotations, string processing, I/O, and so forth.
- **Jetty Server 7.2.0** – Apache License 2.0 or Eclipse Public Licence 1.0
Jetty is a pure Java-based HTTP server and Java Servlet container. Jetty is developed as a free and open source project as part of the Eclipse Foundation.
- **Trove4j 3.0.2** – LGPL 2.1
The Trove library provides high speed regular and primitive collections for Java.
- **Lorem Ipsum For Java** – MIT License
The Lorem Ipsum dummy text generator for Java. (Used in tests.)
- **JSON 20090211** – JSON License (similar to MIT and BSD licences)
- **Apache log4j 1.2.16** – Apache License 2.0
Apache log4j is a Java-based logging utility.
- **Simple Logging Facade for Java 1.6.4** – MIT License
The Simple Logging Facade for Java or (SLF4J) serves as a simple facade or abstraction for

various logging frameworks, e.g. `java.util.logging`, `log4j` and `logback`, allowing the end user to plug in the desired logging framework at deployment time.

- **Hibernate ORM 4.1.0** – LGPL 2.1

Hibernate is an object-relational mapping library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database.

- **Akka 2.0.1** – Apache License 2.0

Actors are very lightweight concurrent entities. They process messages asynchronously using an event-driven receive loop. Pattern matching against messages is a convenient way to express an actor's behavior. They raise the abstraction level and make it much easier to write, test, understand and maintain concurrent and/or distributed systems. You focus on workflow, how the messages flow in the system, instead of low level primitives like threads, locks and socket IO.

- **JOpenID 1.08** – Apache License 2.0

JOpenID is an OpenID 2.0 Java 5 implementation for OpenID sign on.

- **LiftWeb 2.4** – Apache License 2.0

Lift is a free web application framework that is designed for the Scala programming language. It JSON library is used in this project.

- **Weka** – GPL 2.0

Weka (short for *Waikato Environment for Knowledge Analysis*) is a machine learning software that we use to train and apply machine learning models for ranking translation pair candidates.

- **lib-gwt-file** – LGPLv3

lib-gwt-file is a GWT library to add support for the W3C file API and HTML5 file drag-and-drop to GWT. In this project, this library is used for loading local subtitle files.

- **GWT-Bootstrap** – Apache License 2.0

Based on Twitter Bootstrap, GWT-Bootstrap provides simple and flexible components representing Twitter's bootstrap components, styles, and plugins. Used for a user-friendly and pleasant visual design of the GUI.

We also directly use code from other libraries. These are:

- **IdGenerator** from Direct Web Remoting – Apache License 2.0

IdGenerator is a class for generating session IDs.

- **BCrypt** – BSD License

We use BCrypt for saving the passwords to the database.

- **LanguageTools** – LGPL 2.1

Our rule-based sentence splitter is based on LanguageTools' SentenceTokenizer.

Licenses

Most of the licenses are non-permissive and would allow us to publish our code with a non-permissive license, e.g. Apache License 2.0, if we wanted to do so.

However, two libraries are a problem – Weka, that uses GPL, and LanguageTools, which use LGPL, but we use its code directly.

To satisfy the licensing requirements, we need to license our code in a license compatible with GPL 2.0. It would be a very manageable amount of work to replace the two problematic libraries and license the whole project under a non-permissive license.

Chapter 4

Building the Corpus

Since we wanted the translation memory to be functional from the very beginning, we needed to build a parallel corpus – preferably from some repository of subtitles.

Right after we received the data from *opensubtitles.org* and had the “raw” subtitle data, we started to build a parallel corpus from this data to use it for the translation memory. The process of creating the corpus and getting the “clean” translation memory data is described in the following sections.

Since there were more strategies how to process the “raw” data which looked equally promising, we also needed to develop a measure of the quality of the resulting corpus (either by measuring the alignment itself or the resulting corpus).

In this chapter, we discuss these steps, together with a more detailed description of the data we received. We will also discuss various strategies for measuring the quality of the corpus.

4.1 Retrieving Data

There are hundreds to thousands of subtitle files in any major language available on the Internet, and all of those can be easily downloaded individually, file by file. However, for building a bigger corpus, we need the biggest number of subtitles possible – and it is problematic to download from most of the servers in bigger batches (due to the anti-robot protection and so on).

Also, since we wanted to avoid any copyright problems, we would meet by using random subtitles, randomly downloaded from the Internet. Because of that, we directly asked the administrator of the biggest server providing the subtitle site <http://www.opensubtitles.org> for the data. We will refer to the website as OS.org in the following chapters.

We were extremely lucky to receive all subtitle files in Czech and English (but only from media sources, that have subtitles from both of these languages) from OS.org with following license condition (in Slovak):

Titulky mozem poskytnut, s tym ze:

- nebudu sa dalej sirit
- vsade, kde je to mozne a suvisi to s projektom, bude uvedena linka na

`www.opensubtitles.org (stranka programu, dokumentacia, program...)`

Co sa tyka autorskych prav, tak neviem presne ako to je, ale myslim,
ze to je +- ok :)

– English translation:

We can provide the subtitle files under following conditions:

- they won't be provided any further
- a link to `www.opensubtitles.org` will be placed whenever it's possible
(web page of the program, documentation, program itself...)

Considering the copyright law, I am not really sure how it is,
but I think it's ok :)

We decided that this license condition is acceptable for our purposes.

The contents of OS.org is all user-generated, and users of OS.org, when uploading files, agree with a statement where they declare they are holders of all rights to the content they upload, and that they provide the subtitle files as their own intellectual property for public use. Based on this and the license, we think there are no copyright issues.

4.2 The Initial Data Properties

Format

The OpenSubtitles data consisted of the following files:

- file, called `export.txt.gz`, which was a tab-separated table, having information about one subtitle file on one line. The information consisted of:
 - ID of a media source (an internal OS.org ID)
 - name of the subtitle file (a number)
 - language of the subtitle (either czech or english)
 - ID of the subtitle (in the case of split subtitles, the ID is the same, while name of the file is different)
 - information on how many parts the movie was split and which part this subtitles belong to
 - format of the subtitle (*always* srt)
 - description of media source. This description consists of the name of the movie in the case of movies, or of the name of the series and episode in the case of TV shows. What is also present is the year of the movie release.

We found out these data are taken from IMDB (internally, by OS.org) and in almost all the cases determine the movie unambiguously. There are very few cases of two movies with the same name released in the same year, and the probability of both of them having both Czech and English translation is low. We took the name and the year as unambiguously identifying a media source.

There were 676,155 lines in the document (which would imply 676 155 files). There is 15,882 separate media sources in the file (which would imply 42.6 files for one document).

- 139,538 gzipped subtitle files (filenames as described in the table + .gz)

All these files were packed into a tar file, with a size of 3,076 MB.

Cleaning up – non-existent files

As you can see even just by looking at the numbers, the data in the database and the files do not match up exactly. So the initial clean-up was getting only the intersection of files we have and the files that are in the database.

The intersection is 39,712 Czech subtitle files and 97,991 English files of 15,881 media sources – 3,032 MB of zipped data.

Cleaning up – split subtitles

The other step in cleanup is – for simplicity – removing the subtitles that are split to multiple files; we assumed that those split into more parts are probably just split version the complete ones. 81 % of subtitles are in one piece and only 1.7 % movies have subtitles only in multiple files. By deleting subtitles split into more files, we completely lose the Czech side of 64 movies and English the side of 218 movies, which is 3,5% in total.

Cleaning up – Carmencita

There was also one peculiar issue with the media sources. While looking more carefully at the data, we found there were 228 Czech subtitles files, 814 files in total having one particular movie ID and containing absolutely different content. This movie was Carmencita, a 21 seconds long silent film from 1894¹. We concluded this very probably happened due to a server error at OS.org, because the movie has the ID tt0000001 at IMDB. We deleted those as well.

Cleaning up – results

After performing all the mentioned filtering, we had 2,543 MB in 110,312 gzipped subtitle files (32,705 Czech, 77,607 English) of 15,552 movies / TV shows' episodes.

When we did a simple check to find out, how many subtitles are actually in srt format we found out, that from 110,312 subtitles, 432 are not regular srt and might actually be sub (but sometimes, they are, for example, HTML files, or just strange binary data). That means that only

¹See <http://en.wikipedia.org/wiki/File:Carmencita.ogg> for details.

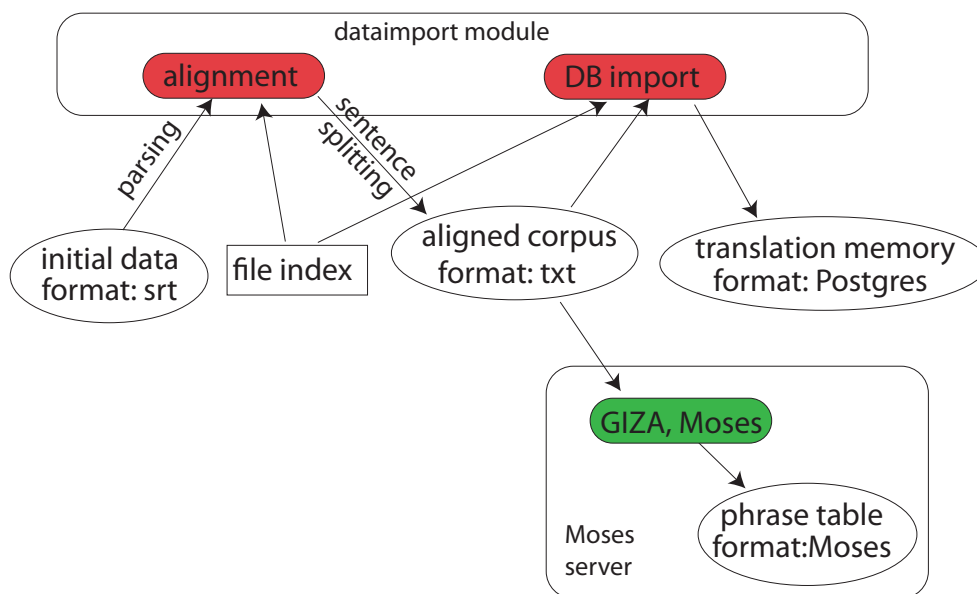


Figure 4.1: How do the data flow from the original archive to the database. We do not talk about the Moses server in this chapter; but it is here to illustrate that the aligned corpus is used in two ways – in building the DB for the translation memory and in building the phrase table for MT.

0.39% of subtitle files are not srt. We are noting it here, because we later based some of our decision based on this

All previous cleanup was done as one of the first things when we received the data; it is not included in the *dataimport* module and the module already expects these data filtered out.

Later in development, we noticed that some files have incorrect language written in the table file; files, that should be in Czech, were in Hungarian, and so on. We added language detection, based on letter trigrams, later; that detection is, however, already in the *dataimport* module and is done directly before aligning the files (which will be described later).

4.3 Loading Subtitles, Parsing

4.3.1 Loading Subtitles

The subtitles were gzipped, as already noted. Gunzipping all the files to disk would take unnecessary space, so we load the files using `GZIPInputStream`, which is a part of the standard Java library.

Most of the Czech subtitles were in Windows-1250 encoding, but a small minority of them were in UTF-8. Some files were not in correct gzip format. Other than these issues, reading the files themselves was quite straightforward.

4.3.2 GUI Parsing

Before we start to talk about parsing, let us take a small sidestep to GUI.

We wanted to move the parsing of the users' uploaded subtitles to the client side. The most important reason was that we did not want to add work to the server. Therefore it was necessary to implement the parsing in the shared classes to make it available both in the GWT based GUI and the dataimport module.

This is, by the way, one of the benefits of having the dataimport module a part of the whole system. We can easily share the code with other parts of the system, and even let it be translated into JavaScript.

However, because of this, we had to write the parsing classes in such a way, that would compile with both GWT and the Java compiler. What that effectively meant was not using the standard Java Regular Expression classes, but special GWT classes.

4.3.3 Parsing

Parsing the subtitles themselves was not a hard task, using GWT's regular expression classes. Even though sub is considerably less popular, we decided to not neglect it and built a parser for it, too.

However, there are two scenarios, depending on when parsing is needed.

One scenario is parsing the movie subtitle after the user uploads it for translation and we want to show it to him. The second scenario is during the phase of building the corpus, where we parse the subtitles before aligning the subtitle items.

Parsing - first step

What both scenarios have in common is the first step. In it, we take the text of the file and get only the subtitle items – we call the subtitle item `UnprocessedChunk` and the first step `UnprocessedParser`.² `UnprocessedParser` is an abstract class, that has two subclasses – `UnprocessedParserSrt` and `UnprocessedParserSub`.

Parsing the files is a relatively straightforward task, even though we have to deal with various edge cases.

Parsing - second step

This is where the two scenarios are different.

When the user uploads a file, as the second step, we take those unprocessed chunks and try to split them to sentences using `TitChunkSeparator` (which, in turn, uses a `SentenceTokenizer`); we speak about sentence splitting in the next chapter. We then take those and return a list of `TimedChunks`, with the appropriate IDs. We proceed to display the chunks using the GUI, which is not described in this chapter.

²As noted in the glossary, we use the word “chunk” in a more general sense in the source code than here in the documentation.

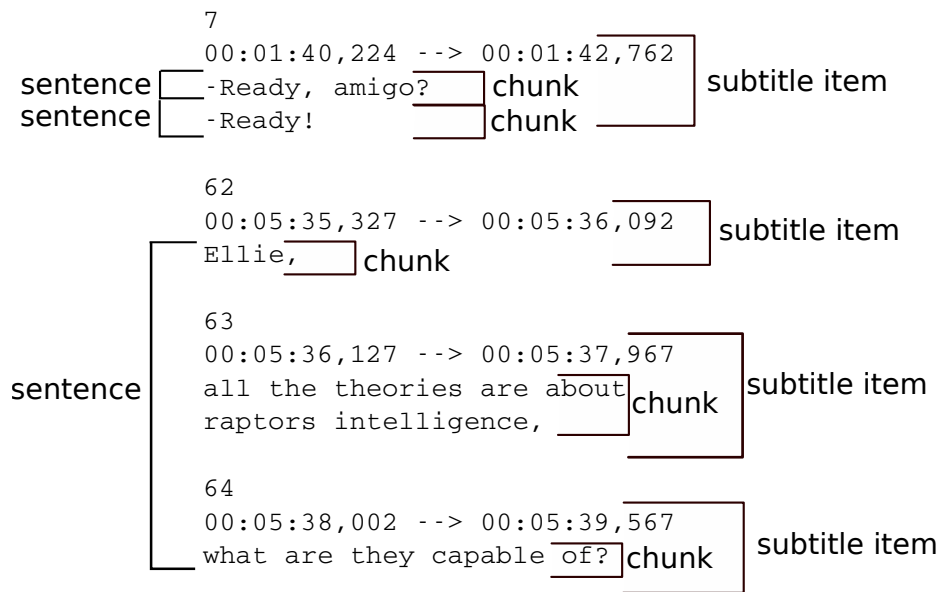


Figure 4.2: Relation of chunk, sentence and subtitle item. From movie *Jurassic Park III*

When we are building the corpus, though, we do not split the subtitle items into sentences right away. We first do alignment of files and subtitle items, which is describe below, and only then split the chunks into sentences, again using `TitChunkSeparator`.

4.4 Sentence Splitting

Basic idea

Quite early on, we decided, that we should not add the whole subtitle items into the translation memory, but that we should split them to smaller parts.

As you can see on Graph 4.2, we decided that we *do* want to break subtitle items into sentences wherever possible and add those to database separately, but we do *not* want to join the sentences that were already broken into multiple items.

One of the reasons for *not* joining sentences together is that it is not trivial to tell if the sentence ends with a given item or not. The other reason is that the user will also have the sentences in his original subtitles split in this way. We would then, for translation, need to join his sentences in the same way - but then, we would have to break them again when exporting out again.

On the other hand, the reason for *not* keeping only the splitting to subtitle items and for splitting into more smaller chunks is, that the splitting into subtitle items seems sometimes too arbitrary and we would miss the shortest sentences, which are the easiest ones to find in a translation memory.

To reiterate: we split sentences only within one subtitle item, and we do not join subtitle items to form complete sentences.

For the same reasons as parsing, we split sentences in the client GUI when the users upload

their subtitles. That limits us in the choice of splitting algorithms, basically to just using regular expressions; any algorithms, based on some machine-learned models, are not applicable here – we cannot link a library to the GWT code and use a model, loaded from a file.

Dialogues

Dialogues are sometimes marked with a dash in the beginning. An example can be seen in Figure 4.2. A dialogue is *always* a new sentence.

Punctuation marks on newlines

While punctuation marks (!?.) do not always end a sentence, they *do* end a sentence when they are right before a newline character. For this reason, we do split sentences on punctuation marks before newlines.

This and the previous rule are implemented in the `TitChunkSeparator`, that is, after applying them, calling `SentenceTokenizer` for rule-based sentence splitting.

Rule-based sentence splitting

Besides the two aforementioned rules, we use rule-based sentence splitting (or Sentence Tokenizing) derived from code from Daniel Naber’s `LanguageTools`.³ The code was heavily edited by us, but only for better clarity and usage of GWT regex classes, the functionality was unchanged. The code is in package `cz.filmtit.share.tokenizers`; the base class is named `SentenceTokenizer`.

The rules are, actually, not *that* sophisticated; they rule out common abbreviations, names of months, cases like `I (really!) did it.`, and so on.

In the rule-based classes, the list of possible words before the punctuation that do not break the sentence have to be explicitly written in the code (because GWT has a problem with reading files). Therefore, if there was a need of adding a new language to `FilmTit`, this would have to be added. On the other hand, similar data are publicly available for every language in, for example, the Wiktionary project.

4.5 Aligning the Subtitles

The most crucial part in building the corpus is to group the subtitles of the same movie together.

We used two different approaches for this. Both of them need to solve these three issues:

- file to file alignment – in each movie find *one* best pair
- filter out the “bad” movies – filter out the movies, where even the best pair of file is not correctly alignable
- item to item alignment – if we already have pairs of files, extract the bilingual pairs of items

Implementation of all techniques described below are in the `cz.filmtit.dataimport.alignment` package.

³<http://www.language-tool.org/>, retrieved 20.08.2012.

4.5.1 File to File Alignment

As you could read in previous sections, we *do* have some mapping from subtitles to movies. Now, we have to find out for each movie, which English and Czech file matches best together; we will throw away the rest of the files except for this one chosen pair for each movie.⁴

What helps us in determining alignment is the fact, that Czech subtitles are usually created by taking existing English subtitles and translating them item by item. The timing of such files should be very similar.

On the other hand, people, who download movies (and, therefore, subtitles) online, often use sources of questionable legality, which usually have more versions of the same file (some of them are so-called “cam-rips”, some of them are copied from DVDs or blue-ray disks). Similarly with TV episodes – depending on the source, the timings are slightly different. Also, in some cases, the subtitle translations are directly copied from DVD subtitles, which have totally different timing.

Sometimes, it also happens that a subtitle file has assigned a completely wrong movie tag.

To illustrate some features of the movie subtitle files, we randomly selected a movie (*Legends of the Fall* from 1994) and we show fragments of *all* its subtitles below. We show 5 lines from beginning, 3 lines from the middle (374 to 376, if possible), 3 lines from the end.

1	00:00:00,066 -->00:00:02,375		
	Titulky přeložené do češtiny by		
	BiGf0oT.		
	Tento disk DVD (Digital Versatile		
	Disc)		
	je určen pouze pro domácí užití.		
	Veškerá práva k obsahové náplni		
	včetně		
	zvukového záznamu přísluší		
	vlastníku		
	autorského práva.		
2	00:00:02,746 -->00:00:05,055	374	
	Neautorizované rozmnožování,	00:56:41,186 -->00:56:44,781	755
	úpravy, projekce pro jiné než	02:01:42,266 -->02:01:47,101	
	domácí	Řekni to ještě jednou	... někde mezi tímto světem
	účely, pronájem, výměna, půjčování	a přestaneme být bratry.	a tím druhým.
	a		
	jakákoli forma přenosu tohoto	375	
	disku	00:56:46,506 -->00:56:49,862	756
	DVD nebo jeho části jsou zakázány.	Někdy!	02:02:23,266 -->02:02:26,178
	Porušování práv vlastníka		Byla to dobrá smrt.
	autorského	376	
	práva bude stíháno podle platných	00:56:52,146 -->00:56:56,503	757
	právních předpisů.	- S tebou nebude šťastna.	02:07:18,306 -->02:07:22,697
		- Uvidíme.	České titulky - BiGf0oT.

⁴Throwing away meant in the sense of building the corpus; we still have the original data, of course.

1 00:00:00,066 -->00:00:02,375 Titulky přeložené do češtiny by BiGf0oT. Tento disk DVD (Digital Versatile Disc) je určen pouze pro domácí užití. Veškerá práva k obsahové náplni včetně zvukového záznamu přísluší vlastníku autorského práva.			
2 00:00:02,746 -->00:00:05,055 Neautorizované rozmnožování, úpravy, projekce pro jiné než domácí účely, pronájem, výměna, půjčování a jakákoli forma přenosu tohoto disku DVD nebo jeho částí jsou zakázány. Porušování práv vlastníka autorského práva bude stíháno podle platných právních předpisů.	3 00:00:32,666 -->00:00:38,059 LEGENDA O VÁŠNI 4 00:00:46,426 -->00:00:51,454 Někteří lidé slyší svůj vnitřní hlas nadměru jasně.	374 00:56:41,186 -->00:56:44,781 Řekni to ještě jednou a přestaneme být bratry. 375 00:56:46,506 -->00:56:49,862 Někdy! 376 00:56:52,146 -->00:56:56,503 - S tebou nebude šťastna. - Uvidíme.	755 02:01:42,266 -->02:01:47,101 ... někde mezi tímto světem a tím druhým. 756 02:02:23,266 -->02:02:26,178 Byla to dobrá smrt. 757 02:07:18,306 -->02:07:22,697 České titulky - BiGf0oT.
1 00:00:46,577 -->00:00:51,605 Somepeople heartheir own innervoices with greatcleamess.	4 00:01:07,137 -->00:01:11,813 Tristan Ludlowwas bom in the moon ofthe falling leaves.	374 00:56:41,337 -->00:56:44,932 You saythat again and we're not brothers. 375 00:56:46,657 -->00:56:50,013 Once! 376 00:56:52,297 -->00:56:56,654 - You know you can't make her happy. - I'm gonna try.	756 02:01:38,937 -->02:01:42,247 He hadalways lived in the borderland, anyway. 757 02:01:42,417 -->02:01:47,252 Somewhere between this world and the other. 758 02:02:23,417 -->02:02:26,329 It was a good death.
1 00:00:58,000 -->00:01:00,696 - I'm here! - Where? I can't see.	4 00:01:35,671 -->00:01:38,037 I got you nov.	315 00:44:59,438 -->00:45:02,032 were married several years ago. 316 00:45:05,278 -->00:45:07,542 Your brother's a congressman now. 317 00:45:09,282 -->00:45:11,716 They have a big, new place in Helena.	
2 00:01:02,704 -->00:01:04,365 I can't move!	5 00:01:40,008 -->00:01:42,272 You're doing good.		
1 00:00:45,869 -->00:00:49,186 <i>Some people hear their own inner voices...</i>	4 00:00:54,157 -->00:00:57,125 <i>Such people become crazy...</i>	374 00:34:22,150 -->00:34:23,710 Charge! 375 00:34:37,734 -->00:34:39,806 Goddamn it! 376 00:34:40,838 -->00:34:42,693 - Where are you hit? - It's just a scratch.	983 02:01:38,444 -->02:01:41,891 <i>He had always lived in the borderland anyway:</i> 984 02:01:41,964 -->02:01:45,576 <i>somewhere between this world and the other.</i> 985 02:02:23,086 -->02:02:25,093 <i>It was a good death.</i>
2 00:00:49,261 -->00:00:50,920 <i>with great clearness...</i>	5 00:00:57,198 -->00:00:59,369 <i>or they become legends.</i>		
3 00:00:50,989 -->00:00:54,087 <i>and they live by what they hear.</i>			
1 00:00:44,411 -->00:00:47,869 Some people hear their own inner voices	4 00:00:53,053 -->00:00:56,147 Such people become crazy	374 00:35:27,326 -->00:35:30,921 Is that wrong to want to distinguish myself gloriously 375 00:35:30,996 -->00:35:33,226 in combat as my father did? 376 00:35:33,298 -->00:35:36,290 Tristan and Alfred watch over me so carefully	1000 02:06:50,103 -->02:06:53,869 Somewhere between this world and the other 1001 02:07:32,979 -->02:07:35,072 It was a good death 1002 02:12:47,200 -->02:12:49,072 {{{the end}}}}
2 00:00:47,948 -->00:00:49,677 with great clearness	5 00:00:56,223 -->00:00:58,487 or they become legend		
3 00:00:49,750 -->00:00:52,981 and they live by what they hear			

1 00:00:43,411 -->00:00:46,869 Some people hear their own inner voices		374 00:35:26,326 -->00:35:29,921 Is that wrong to want to distinguish myself gloriously	1000 02:06:49,103 -->02:06:52,869 Somewhere between this world and the other
2 00:00:46,948 -->00:00:48,677 with great clearness	4 00:00:52,053 -->00:00:55,147 Such people become crazy	375 00:35:29,996 -->00:35:32,226 in combat as my father did?	1001 02:07:31,979 -->02:07:34,072 It was a good death
3 00:00:48,750 -->00:00:51,981 and they live by what they hear	5 00:00:55,223 -->00:00:57,487 or they become legend	376 00:35:32,298 -->00:35:35,290 Tristan and Alfred watch over me so carefully	1002 02:12:46,200 -->02:12:48,072 {{{the end}}}
1 00:00:46,418 -->00:00:51,446 Some people hear their own inner voices with great clearness.		374 00:56:58,898 -->00:57:02,208 You will fail.	752 02:01:42,258 -->02:01:47,093 Somewhere between this world and the other.
2 00:00:51,618 -->00:00:54,655 And they live by what they hear.	4 00:01:06,978 -->00:01:11,654 Tristan Ludlow was born in the moon of the falling leaves.	375 00:57:14,298 -->00:57:17,529 I'm going to be leaving today.	753 02:02:23,258 -->02:02:26,170 It was a good death.
3 00:00:54,818 -->00:01:00,814 Such people become crazy, or they become legends ...	5 00:01:11,818 -->00:01:14,173 It was a terrible winter.	376 00:57:26,458 -->00:57:30,087 I do wish you both all the best.	754 02:07:18,298 -->02:07:22,689 English subtitles - 1FT
1 00:00:45,977 -->00:00:51,005 Some people hear their own inner voices with great cleanness.		374 00:56:40,737 -->00:56:44,332 You saythat again and we're not brothers.	756 02:01:38,337 -->02:01:41,647 He had always lived in the borderland, anyway.
2 00:00:51,177 -->00:00:54,214 And they live by what they hear.	4 00:01:06,537 -->00:01:11,213 Tristan Ludlow was bom in the moon of the falling leaves.	375 00:56:46,057 -->00:56:49,413 Once!	757 02:01:41,817 -->02:01:46,652 Somewhere between this world and the other.
3 00:00:54,377 -->00:01:00,373 Such people become crazy, or they become legends ...	5 00:01:11,377 -->00:01:13,732 It was a terrible winter.	376 00:56:51,697 -->00:56:56,054 - You know you can't make her happy. - I'm gonna try.	758 02:02:22,817 -->02:02:25,729 It was a good death.
1 00:00:43,785 -->00:00:47,243 Some people hear their own inner voices...		374 00:35:35,741 -->00:35:37,470 I may never get the opportunity.	987 02:06:45,772 -->02:06:49,367 He had always lived in the borderland anyway:
2 00:00:47,322 -->00:00:49,051 with great clearness...	4 00:00:52,427 -->00:00:55,521 Such people become crazy...	375 00:35:46,085 -->00:35:47,712 Charge!	988 02:06:49,443 -->02:06:53,209 somewhere between this world and the other.
3 00:00:49,123 -->00:00:52,354 and they live by what they hear.	5 00:00:55,597 -->00:00:57,861 or they become legends.	376 00:36:02,334 -->00:36:04,495 Goddamn it!	989 02:07:32,319 -->02:07:34,412 It was a good death.

As can be seen, the two Czech subtitle files are exact copies of each other. Also, there is one English subtitle that is totally wrong. Except for this one, most of the English files seems to match the Czech one quite well. None of them is a “perfect” match, looking either at numbers or at the timing. Judging by the *number* of the last translation – some are from a similar source (those with around 750 chunks), while those with significantly more chunks are probably from different source. Although, there is no perfect match, there are few almost-perfect matches.

On average, there is 2.1 Czech subtitle files and 5 English subtitle files per movie. However, the numbers are not evenly distributed and very popular movies can have tens of subtitles.

If we take for *each* movie *all* the pairs, where Czech is one side and English on the other (but both have the same movie), we have 235.825 such pairs (reminding, we originally had 139.538 files).

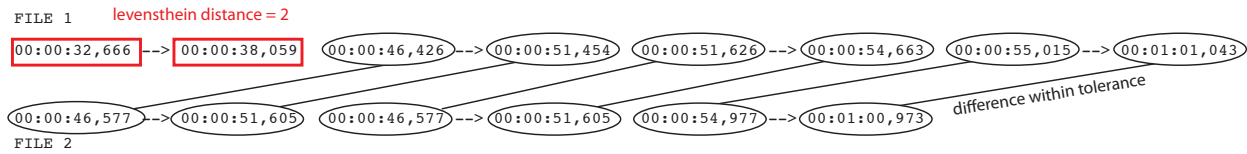


Figure 4.3: Levenshtein distance for file-to-file alignment

4.5.2 Subtitle to Subtitle Alignment

As written above, translations of subtitles are usually done directly from other subtitles, so the alignment is entirely based on time marks. We do *not* look at any other features.⁵

However, even in the cases when a Czech subtitle file has been made directly from an English one, there are still cases where, for example, one chunk is split into two chunks, two are merged into one and so on. Sometimes, the English version of the subtitles is done for people with hearing impairment and, therefore, describes all the sounds, while the Czech version has those deleted.

The result of this step is a list of pairs of the Czech and English subtitle items. We do this step only on the chosen pair of files from the previous step, and only if the pair of the files is chosen as a “good” candidate.

After this step, we almost have the final corpus. What we have to do before saving it into a final corpus is splitting the aligned items into chunks, as described in Section 4.4. If both Czech and English have the same number of chunks, we split them and add those to the corpus; if the number is different, we ignore them.

4.5.3 Approaches

Equality with tolerance

The first approach is to take only the items with the same time as belonging to each other.

However, the times are rarely exactly equal. One of the paper authors⁶ used 0.6 s as a tolerance in order not to be confused by slight differences in timing.

We observed that if we use a bigger tolerance, we have more subtitle pairs, but sometimes with worse quality (because sometimes we get totally different subtitles, not belonging to each other). We tried tolerance 0.6 s (as suggested by the paper), and then 6 s.

For counting file to file alignment, we count editing distance of their time marks, as illustrated in Figure 4.3; we use a dynamic algorithm for that.

We take all the time information (both the starts and the ends) as vectors. Then, we count editing distance of those two vectors, where we take the two time informations as equal if they fall within tolerance, otherwise we take them as unequal. We score all possible pairs of Czech

⁵In *Using Movie Subtitles for Creating a Large-Scale Bilingual Corpora* (Einav Itamar, Alon Itai), authors tried to use other features, but we used just the time info.

⁶Einav Itamar, Alon Itai (2008): *Using Movie Subtitles for Creating a Large-Scale Bilingual Corpora*. The sixth international conference on Language Resources and Evaluation, LREC 2008.

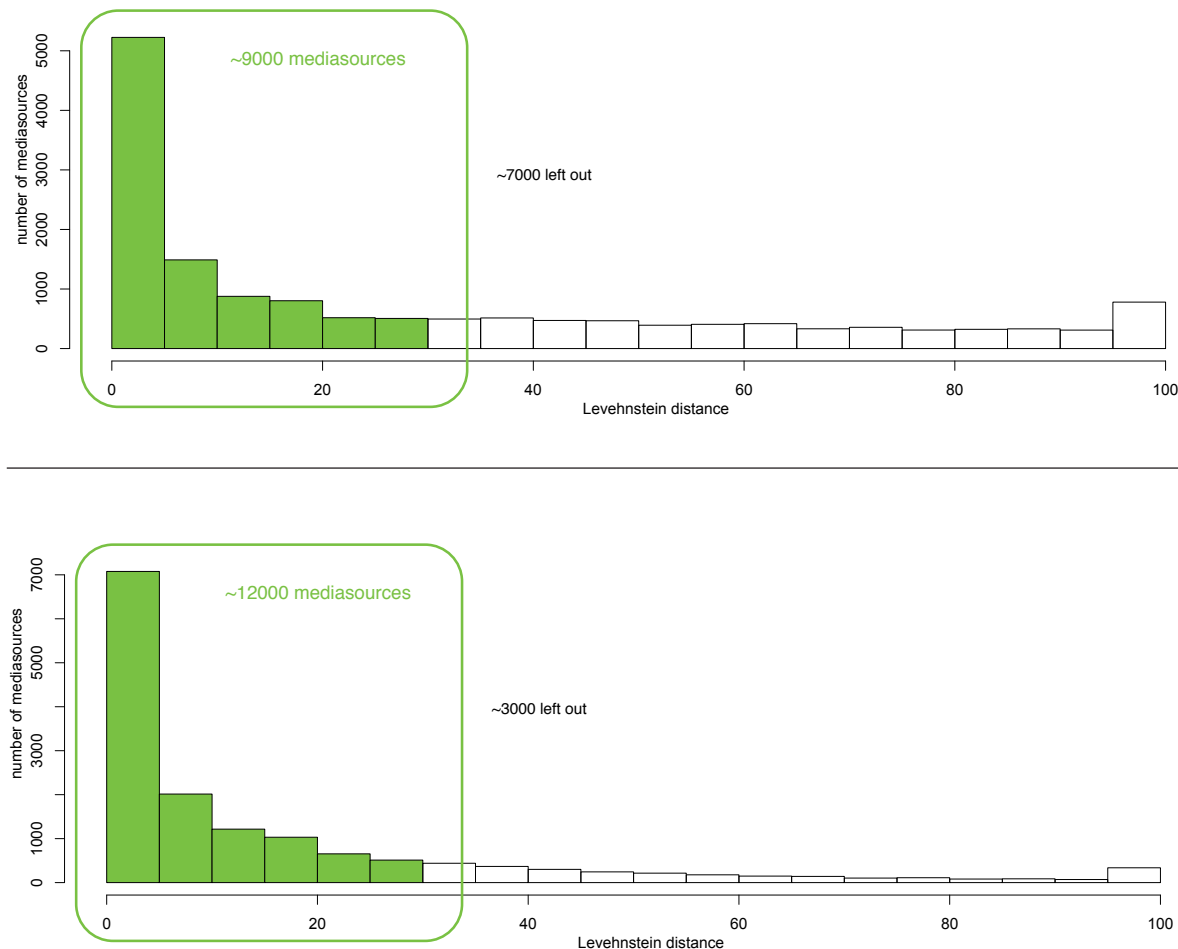


Figure 4.4: Levenshtein distances for 0,6s tolerance (above) and 6s tolerance (below)

subtitle files on one side and English on the other for the given movie and we use the Levenshtein distance as a score; the file pair with the shortest Levenshtein distance is the “chosen” file pair.

We use Levenshtein distance because there can be some small splitting, additions, and so on, but we want to take the pair of files, that have minimal number of these.

For filtering out the “bad” movie pairs, we just throw away all that has editing distance higher than 30. (This was decided empirically by looking at some random subtitle files.)

For subtitle to subtitle alignment, we take the subtitles that have a distance of both beginning and ending within the tolerance. Because the computation on the whole files would be very time consuming, we limited it to just the first 100 time marks in the files.

You can see the histogram of shortest Levenshtein distances for each movie in Figure 4.4. The green part represents the “good” file pairs. The distance is maximally 100, since we take only the first 100 subtitles from each file. You can already see that there are more files in the bigger

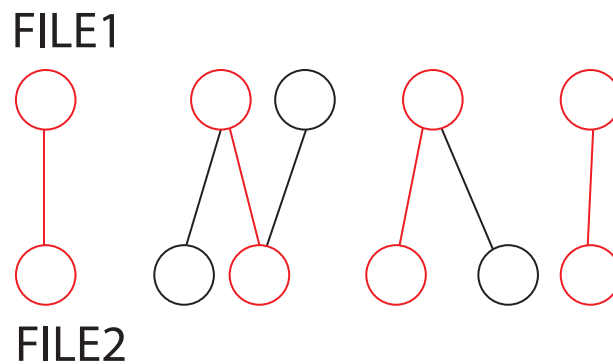


Figure 4.5: An illustration for the distance-based alignment. Circle represents subtitle, red line represent chosen alignment, black ones are some candidates, red circles are the subtitles that are aligned

tolerance. (Of course, they might be also more wrong.)

The chunk-to-chunk alignment is then simple – chunks are aligned together if both the time of their start and time of their end differ less than by the tolerance.

Shortest distance

The idea that lead to the next approach was – what if we didn’t have any maximal tolerance and just took the subtitles that are closest to each other.

More concretely, we took the middle time of each chunk (in milliseconds) and counted distance from other chunk as an absolute difference of these middles.

To find the best matching pairs in pair of two files, we took the closest Czech chunk for each English chunk; then, if more English chunks were aligned with the same Czech chunks, we select the closest.

(Although similar, this is not a case of graph matching in bipartite graphs, since it is *not* needed that each Czech chunk would have some English chunk and vice versa.)

Trivially implemented this algorithm would be quadratic to the size of subtitle file; however, we may dynamize it so even the worst case has linear time to the size of the subtitle file.

In the file-to-file alignment case, we count this alignment for each pair of files and then we sum all the distances for a given pair of files; we then select the pair with the smallest sum as the correct one. If the subtitles are just translated from each other, the sum should be small and can be even zero, if the time marks are exactly the same.

We then selected given number of movies (we tried 6,000 and 12,000) with the smallest sum as the “good” movie pairs and then align them.

Baseline – trivial alignment

As a baseline for comparison, the “trivial” algorithm just takes first filepair that it sees for a given movie, then takes all the movies, and then aligns first chunk on source side with first chunk on

target side, second with the second one and so on.

4.5.4 Evaluation of Alignments

We have two slightly different approaches to counting the alignment of subtitles. The question is, which one to choose. For having a more rigorous evaluation than just manual comparing of random files it was necessary to develop a metric which captures the quality of the alignment which is described later.

Size of corpus

One of the most simple evaluations is just the sheer size. Since the alignment classes are modular, we can actually use different file-to-file alignment together with subtitle-to-subtitle alignment.

file alignment	subtitle alignment	number of pairs
Tolerance 0,6 s	Tolerance 0,6 s	5.412.875
Tolerance 6 s	Tolerance 0,6 s	7.210.505
Tolerance 6 s	Tolerance 6 s	9.228.304
Tolerance 6 s	Distance	4.660.382
Distance 6.000	Distance	2.109.017
Distance 12.000	Distance	4.177.117
Distance 6.000	Tolerance 6s	4.024.876
Distance 12.000	Tolerance 6s	7.982.142
Trivial	Trivial	8.318.225

Table 4.1: Size of corpus

As can be seen in Table 4.1, the highest tolerance produces the biggest corpus, while Distance-based alignment does not produce as much.

What is interesting is that the Trivial alignment – that really takes *everything* – has smaller corpus than the one with the higher tolerance.

Manual alignment

Another measurement is to compare the alignment with a human alignment.

What is sometimes proposed in cases on alignment evaluation is letting users align something and then measure that against the aligner output.

The problem is – what to manually align exactly in our case? We can manually align *files* for a given movie (we have shown an example of all files for a movie in 4.5.1). It is not very easy though, since – as we have seen already – the files are all *more or less* correct translations and the timings are not exact matches. Nonetheless, we randomly selected 30 movies and for each of these, selected which pairs of *files* seems like from the same source and match together.

However, there is a bigger issue with the manual chunk-to-chunk alignment, that is – which file pairs to manually align? Since we do not know the file-to-file aligner’s results in advance, we

should align all the file pairs – but if there are, for example, 10 subtitle files on each language, it produces hundreds of pairs of files on which we should provide manual alignment, while the texts in all of them are *mostly* the same, but *not exactly* the same. This actually proved too hard and tedious to do.

Moreover, the aligner chooses just one pair, but a person would have to align all the possible pairs for no reason (a person can also produce different errors in different file pairs, which could be “unfair” to some of the aligners).

Therefore, we manually annotated just the file-to-file alignment and we tried a different metric on the whole chunk-to-chunk alignment.

Counting precision and recall for file to file alignment

There are five cases of possible events:

- With manual alignment, we found no possible matches and the aligner did not mark it as a “good” movie - in this section, we call this *true negative*
- With manual alignment, we found possible matches, but the aligner did not mark it as a “good” movie - in this section, we call this *false negative*
- With manual alignment, we found no possible matches, the aligner marked it as a “good” movie and included it in corpus - in this section, we call this *false positive*
- With manual alignment, we found possible matches, the aligner marked it as a “good” movie, but the match from aligner was not one of the correct ones - we call this *partly true positive*
- With manual alignment, we found possible matches, the aligner marked it as a “good” movie and the match from aligner was one of the correct ones - we call this *fully true positive*

With some consideration, we count precision as ⁷

$$\frac{\text{fully true positive}}{\text{false positive} + \text{fully true positive} + \text{partly true positive}}$$

and recall as

$$\frac{\text{fully true positive}}{\text{fully true positive} + \text{partly true positive} + \text{false negative}}$$

This means precision is the percentage of movies chosen by the aligner that are aligned correctly; recall is the percentage of movies that *can* be aligned (from manual alignment), that are aligned correctly. Results for the algorithms are in Table 4.2.

⁷Strictly speaking, it is not precision or recall. However, intuitively it captures similar qualities and it may be more confusing to use a different terminology.

approach	precision	recall	F1-measure
Baseline	33	33	33
Tolerance 6s	61	57	59
6,000 best distance-based	67	20	31
12,000 best distance-based	76	53	63

Table 4.2: Performance of the alignment algorithms (as percentage)

Simulation of translation memory

We tried a different evaluation metric for testing the alignments.

We imported the whole corpus (as created by an aligner) except for 30 randomly selected media sources into a database, then we took 20 chunks from each of the 30 files and queried it to a translation memory. We looked at the result this translation memory has given us and judged how much of it was relevant.

We did not test it on the real database (since building it multiple times would be too time consuming), we searched just for direct matches using standard Java⁸ “equals”.

Translation memory simulation result

We counted two percentages for each configuration – the one we called *precision* – that is, how many of the received translation results were correct – and the second that we called *coverage* – that is, *how many of the subtitle chunks received at least one good translation*.

We also calculated a harmonized mean of those two.

What also has to be noted here is that the result counts are not evenly distributed – for the more frequent sentences like “I don’t know”, we might have hundreds of matches; we receive only the first 30 matches. This will, of course, skew the precision slightly (since we take only 30 chunks as received); however, it still emulates the working of translation memory.

Results of the experiment are tabulated in 4.3.

file alignment	subtitle alignment	precision	coverage	mean
Tolerance 0,6 s	Tolerance 0,6 s	96	19	31
Tolerance 6 s	Tolerance 6 s	82	29	43
Tolerance 6 s	Distance	83	24	37
Distance 6.000	Distance	89	21	35
Distance 12.000	Distance	85	24	38
Distance 6.000	Tolerance 6s	79	24	36
Distance 12.000	Tolerance 6s	78	27	41
Trivial	Trivial	15	20	17

Table 4.3: Size of corpus

⁸Standard Java, but it’s in Scala

It can be noticed that the Distance is always worse than Tolerance. Also, we might notice, that every time we somehow allowed more files, the precision went down, but the coverage went up.

The baseline algorithm is absurdly tolerant and produces actually bigger coverage than the tolerance-based algorithm with tolerance 600 ms. However, the precision is very low.

Also note that even with the alignment with best coverage, we still covered only slightly more than one fourth of the file. We see that we really *need* to use fuzzy matching and, if possible, machine translation, both of which we use in the final software.

We chose the tolerance-based algorithm with 6s tolerance, since it has the best harmonic mean and mainly has the biggest coverage with still relatively precise translations. Users of Machine Translation can easily ignore a possible wrong translation, if they do not like it.

4.6 Running the Alignment

The alignment can also be re-run. The needed Scala objects with appropriate main methods are in `cz.filmtit.dataimport.alignment.tasks` package. However, it is has not been tested on any other data than our data. The locations of the initial data, the `export.txt` file and the directory where the corpus will be located has to be set in `configuration.xml`.

Alignment with any of the methods takes about 4 hours on a Linux PC with Intel Core2 Quad CPU with 2.83GHz with 4GB of memory.

Also, the tests that produced the results in this section are in the same package. However, as they were intended to be run only once, they may have paths and options hardcoded in the source code.

Chapter 5

Core Translation Memory

This chapter describes the design and implementation of the core translation memory. The core translation memory is the part of the project that is responsible for retrieving and ranking suitable translation suggestions for chunks in the subtitle file that the user is translating.

5.1 Preliminaries

5.1.1 Choosing a DBMS

For the choice of a suitable database management system underlying the core translation memory and the user space, the main points we considered were

- software license of the DBMS
- general performance and maintainability
- included support for fuzzy matching and custom indexes

According to these requirements, we evaluated several database systems and selected the open source database system Postgres.¹ This system fulfills the requirements as follows:

- open license similar to BSD license
- good results in performance evaluations and good reputation for maintainability
- support for phonetic representation of strings (SOUNDEX and METAPHONE), string edit distance (Levenshtein), fuzzy string search using character trigrams and customizable indexes (GIST, GIN)

For managing connections to the database in Java and Scala, we use JDBC, which would allow the database connection to easily use another DBMS. In the XML-based configuration file (see the installation manual, Chapter 11), the JDBC connector has to be specified, together with

¹<http://www.postgresql.org/>, written either Postgres or PostgreSQL

the username and password to the database. When testing the translation memory with project-internal unit tests, we replace the production JDBC configuration with a temporary in-memory database (HSQLDB²).

```
1 <database>
2   <connector>jdbc:postgresql://localhost/filmtit</connector>
3   <user>postgres</user>
4   <password>postgres</password>
5 </database>
```

Listing 5.1: Configuration snippet

5.1.2 Retrieving Media Source Information

For retrieving information about media sources, we initially used a free IMDB API (<http://imdbapi.com/>). However, it showed that this API might become unavailable in the future and indeed, as of July 30, the API had been shut down. To avoid complications early, we switched to the Freebase API.³ Freebase is a large knowledge base of structured data based on data extracted from Wikipedia and other publicly available resources and continually extended by users.

To retrieve media source candidates, first a full text search query with the title and a restriction to entities of type *film*⁴ and *TV program*⁵ is sent to the Freebase API. Afterwards additional information for each candidate is retrieved and the `MediaSource` objects are sent to the GUI as media source suggestions for the user to select the best one.

5.2 Architecture of the Core Translation Memory

The core translation memory consists of the database of chunks and media sources (movies and TV shows) which are indexed in several ways for fast retrieval. A query to the translation memory generally proceeds in two steps: In the first step, translation pairs for a chunk are retrieved from the database. The second step consists of ranking the candidates retrieved in the first step according to their quality and how well they match the query. If the quality of the retrieved candidates exceeds a minimum quality threshold, they will be sent to the user.

5.2.1 Backoff Translation Memories

The idea of a backoff translation memory is to use multiple ways of retrieving and ranking candidate translation pairs from the database and “back off” to a less exact level of retrieval and ranking when there are no satisfying results on the current level.

A backoff-level in the translation memory consists of a *translation pair searcher*, a *translation pair ranker* and a *minimal quality threshold*. If the results retrieved by the *searcher* and scored and

²<http://hsqldb.org/>

³<http://www.freebase.com/>

⁴Freebase type: /film/film.

⁵Freebase type: /tv/tv_program.

ranked by the *ranker* do not meet the threshold, then the query is sent to the next level of the backoff translation memory.

5.3 Candidate Retrieval

We implemented several methods for efficiently retrieving candidate translation pairs from the database.

5.3.1 Translation Pair Searchers

Every translation pair searcher implements the interface `TranslationPairSearcher` and therefore must implement the method

```
1 def candidates(chunk: Chunk, language: Language): List[TranslationPair]
```

Listing 5.2: Candidate search

This is the main method for retrieving translations pairs, which are then combined and ranked in later steps. A `TranslationPairSearcher` may require the chunks it queries to be tokenized beforehand. In this case, the `TranslationPairSearcher` must override the following method:

```
1 def requiresTokenization: Boolean
```

Listing 5.3: `TranslationPairSearchers` may require tokenization

`TranslationPairSearchers` that are based on the local Postgres database extend the class `TranslationPairStorage` and must implement the method `add` for adding new translation pairs to the storage.

```
1 def add(translationPairs: TraversableOnce[TranslationPair])
```

Listing 5.4: `TranslationPairSearchers` may require tokenization

The method `reindex` must be implemented to create and update the database indexes for the particular searcher.

```
1 def reindex()
```

Listing 5.5: `TranslationPairSearchers` may require tokenization

5.3.2 Tokenization

For all program-based retrieval methods, we tokenize all inputs, i.e. we separate each input sentence into individual tokens (typically words). We use the tokenizers provided by the Apache

OpenNLP project.⁶ OpenNLP provides both unsupervised and supervised methods of tokenization. As the default tokenizer, we use an unsupervised tokenizer that splits the input based on whitespace. This method has the advantage that it is applicable to any new language without requiring further training, however the result is of moderate quality and often leads to problems in latter processing steps (e.g. Named Entity Recognition).

To improve the results, we use a Maximum-Entropy based OpenNLP tokenizer. For English, we use the models provided by the OpenNLP project.

Czech ME tokenizer

For Czech, no such model existed, so we trained a Maximum Entropy tokenizer model on data from the Prague Dependency Treebank.⁷ Scheme of the process is in figure 5.1.

We produced a script that creates training data in the following format:

K vynikajícím spisovatelům<SPLIT>, kteří věřili v nevysvětlitelné duševní úkazy<SPLIT>, patřil také americký romanopisec Upton Sinclair<SPLIT>.

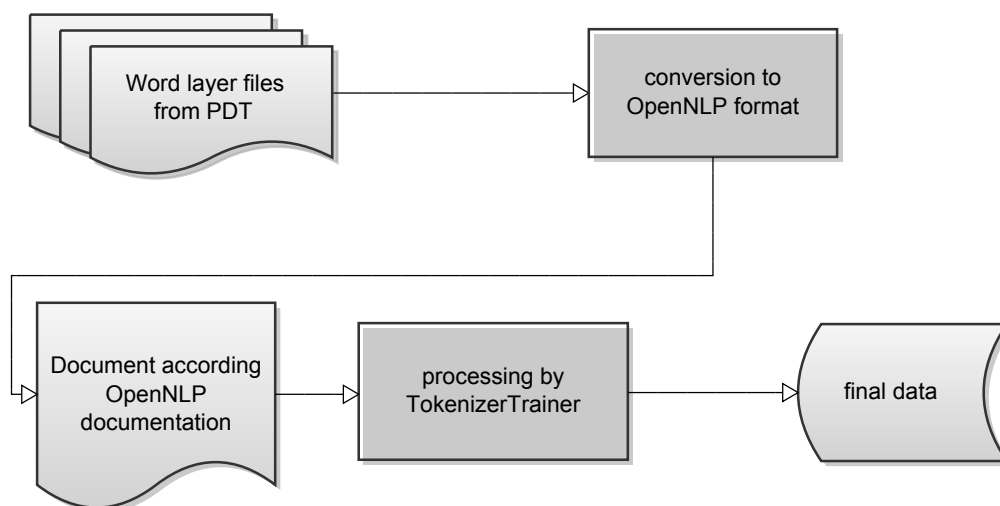


Figure 5.1: Scheme of training the tokenizer.

In the training data, the <SPLIT> annotation indicates a position in the data, in which the tokenizer is required to separate two tokens (in this case words and punctuation symbols). These positions were manually marked in the Prague Dependency Treebank by annotators.

On a test dataset from the Prague Dependency Treebank, the ME tokenizer achieved the following results:

- 1 Precision: 0.99702
- 2 Recall: 0.99547
- 3 F-Measure: 0.99624

⁶<http://opennlp.apache.org/>

⁷<http://ufal.mff.cuni.cz/pdt2.0/>

Listing 5.6: Accuracy for the Czech ME tokenizer.

5.3.3 Signature-Based Retrieval

The most general method is the usage of indexed signature strings. There are two implementations of signature-based retrieval in the project: In the first method, signatures are created fully within the Postgres database (database-based signatures) and in the second, the signatures are created on the server and only stored and retrieved in the database (program-based signatures).

Program-based signatures

In the case of program-based signatures, for each chunk stored in the database, a string is produced by a signature function and this string is used to retrieve candidates from a signature table that is in turn connected to the translation pairs table.

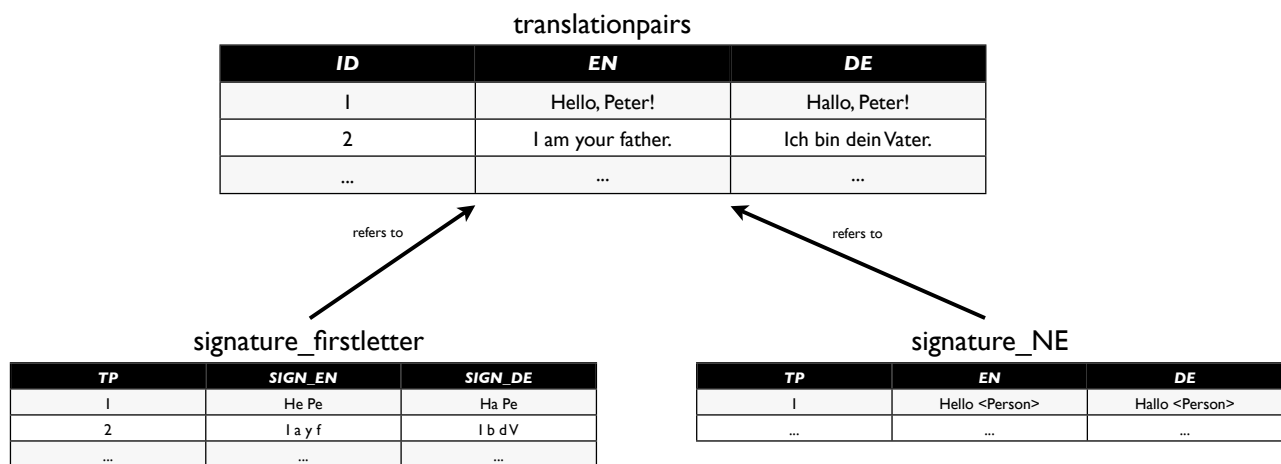


Figure 5.2: Illustration of signature strings in the database.

Database-based signatures

In the case of the database-based signatures, a PL/pgSQL procedural function is used to create a functional index⁸ on the translationpairs table. The function can then be used for fast retrieval from this table.

Several signature functions are used in the translation memory implementation. In the following sections, the signature functions and related issues will be illustrated.

⁸See e.g. <http://www.postgresql.org/docs/9.1/static/indexes-expressional.html> (retrieved 02.08.2012).

5.3.4 Signature-Based Retrieval: Exact Matches

The first signature function is for retrieving exact matches. For this, the signature function consists of the first letters of each word in the chunk. Punctuation is dropped from the signature. If the queried chunk is short, only using the first letter would produce a very high number of results. Hence, for short chunks, more letters of each individual token are included. Algorithm 1 shows pseudo-code for the FirstLetter function.

```

Data: text chunk
Result: signature for the chunk
tokens ← TokenizeAndRemoveStopWords(chunk)
for token ∈ tokens do
  if |tokens| = 1 then
    | ret += Lower(token)
  else if |tokens| = 2 then
    | ret += Lower(Take(3, token))
  else if |tokens| = 3 then
    | ret += Lower(Take(2, token))
  else
    | ret += Lower(Take(1, token))
  end
end
return ret joined by " "

```

Algorithm 1: FirstLetter function.

5.3.5 Signature-Based Retrieval: Named Entities

The second, more fuzzy signature function uses named entity recognition to provide matches. Named entity recognition is the task of finding elements in a text belonging to basic name categories like *Person*, *Organization* and *Place*. In the signature, the surface form of the named entity is replaced by its named entity category. This allows the retrieval of candidate chunks that differ only in the named entities they are using. As an example, consider the following chunks:

```

1 Chunk 1: Peter saw the girl.
2 Signature: <Person> saw the girl
3
4 Chunk 2: Thomas saw the girl.
5 Signature: <Person> saw the girl

```

Listing 5.7: Example of NE-based retrieval.

Using the named entity-based signature, Chunk 2 can be retrieved as a candidate for Chunk 1. Since an exact match would be preferable, this will only be used if there is no better fitting candidate in the database.

Named Entity Recognizers

For Named entity recognition, we use the OpenNLP Maximum Entropy based Named entity recognizers. For English, we were able to use the pre-trained models for persons, organizations and places that are available through the OpenNLP project.

For Czech, no such models were available, hence we trained our own models based on various data sources and then chose the best models based on a comparison of two approaches of acquiring the training data.

First approach: Training data based on Wikipedia and DBpedia

Our first approach to acquiring named entity recognition training data for Czech was based on the Pig NLProc utility.⁹ Pig is an Apache project providing platform that features an SQL-like syntax for data analysis based on Apache Hadoop. The Pig NLProc utility provides a number of Pig scripts to extract natural language processing training data from Wikipedia and DBpedia.

DBpedia¹⁰ is a community project for extracting structured data from Wikipedia. Based on user-created mappings of Wikipedia info boxes to an ontology, DBpedia provides data that allows to easily select DBpedia entities that correspond to persons, organizations, places, etc.

To acquire named entity recognition training data, the Pig NLProc utility searches for article references to Wikipedia pages that are known (from the DBpedia ontology) to be of the required types (e.g. Person). These instances are sentences with links to the given article and these are then converted to the training format required by OpenNLP.

For Czech, user-generated mappings for many info boxes are already available, however, the DBpedia data for Czech is not yet available. Hence, we downloaded the DBpedia extraction framework with the Czech data and ran the extraction process ourselves. With the resulting data and the Pig NLProc scripts (which required minimal changes to resolve compatibility issues), we produced training data for named entities of the types Person, Place and Organization.

Second approach: Training data based on the Czech Named Entity Corpus

The second approach is based on the Czech Named Entity corpus.¹¹ The corpus contains manual named entity annotations of a large number of named entity types. Figure 5.3 shows an overview of the NE types annotated in the corpus.

We wrote scripts that convert the XML-based format of the corpus to the format required by OpenNLP, manually selected the annotation types we were interested in and trained the OpenNLP named entity recognizers on the resulting data.

Selecting the best approach

Both approaches showed to have distinct advantages and disadvantages. While the first approach is only minimally supervised and can easily be extended to other languages that have a local

⁹<https://github.com/ogrisel/pignlproc>

¹⁰<http://dbpedia.org/About>

¹¹http://ufal.mff.cuni.cz/tectomt/releases/czech_named_entity_corpus_10/index.html

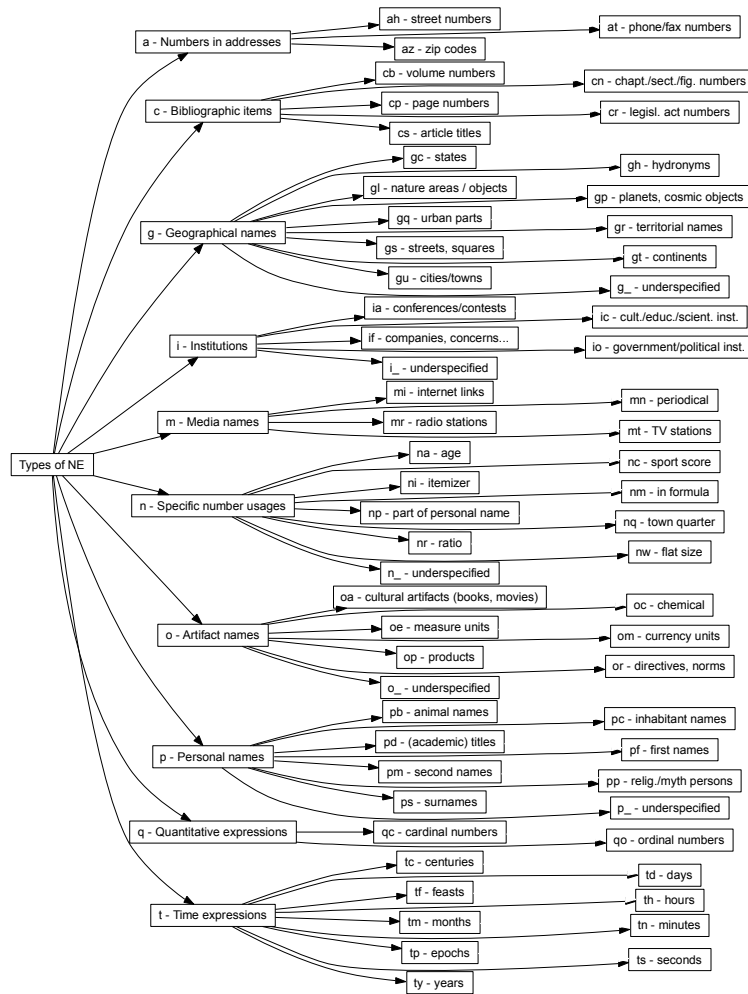


Figure 5.3: Named entity types in the Czech Named Entity corpus. Source: Czech Named Entity corpus documentation.

version of Wikipedia and existing DBpedia dumps (DBpedia dumps are currently available for 97 languages¹²), our manual evaluation of the Named Entity recognizers produced showed that the models trained on the Czech Named Entity corpus produced better results.

More precisely, the DBpedia recognized far less cases and, therefore, had much lower RECALL, with its PRECISION being only slightly better. We then decided that lower RECALL is worse for our purposes – it is better to show a slightly wrong translation to the user than not to show anything.

¹²<http://wiki.dbpedia.org/Downloads37>

Effectiveness

It is worth noting that the overall effectiveness of using named entity-based retrieval was worse than we anticipated. In most subtitle files, there are only few matches, but the recognition of named entities takes the most time during the data import. For this reason, we are not currently using it; the other methods already provide reasonable results, are faster and are easier to adapt to new languages.

5.3.6 Full-Text Search

As another, more fuzzy level of retrieval, we use the full-text search included in Postgres. In Postgres full-text search, documents (in our case translation pairs) are tokenized and normalized. The result of this process is a vector of normalized tokens (`tsvector`). The normalization step can include lowercasing, stemming, using synonyms from a dictionary or removing stop words. Postgres builds an index over the token vector of all documents, against which queries can be run. To convert a string to a query, it is tokenized and normalized and the resulting normalized tokens are connected by an AND operator.

The tokenization and normalization for a specific language is described in a text search configuration.¹³ By default, Postgres includes text search configurations for the following languages: Danish, Dutch, English, Finnish, French, German, Hungarian, Italian, Norwegian, Portuguese, Romanian, Russian, Spanish, Swedish, Turkish and the default “Simple” configuration. For Czech, a text search configuration is available from postgres.cz.¹⁴ For installation instructions, please see the technical manual in Chapter 11.

When querying candidates for a Chunk, we use the `ts_rank`¹⁵ function to pre-sort the candidates and the `ts_rank` score is later used as one of the scores in the ranking of translation pair candidates. One problem that the full-text search as a method of candidate retrieval shows, is that some of the candidates are unreasonably large documents that still have high scores because their token overlaps with the query are reasonable. We approach this problem by using our own ranking functions for translation pairs, as described in Section 5.4.

5.3.7 External Services

As another backoff level, we use Machine Translation from external services. For this, a query with the chunk is sent to an external REST-based API. We offer a translation pair searcher for the free Machine Translation API of the MyMemory project.¹⁶ However, there is currently no machine translation service with an unlimited free API, and, for example, the MyMemory project has such a low daily quota (2500 requests/day) that the quota is quickly surpassed when translating subtitle files.

¹³<http://www.postgresql.org/docs/9.1/static/textsearch-intro.html>

¹⁴http://postgres.cz/wiki/Instalace_PostgreSQL

¹⁵<http://www.postgresql.org/docs/9.1/static/functions-textsearch.html>

¹⁶<http://mymemory.translated.net/doc/spec.php>

5.3.8 Statistical Machine-Translation Based on Moses

We tried Machine Translation first as an experiment, but the results turned out to be very good and, after some optimizations, also provided very quickly.

Moses¹⁷ is a very popular statistical machine translation system that can work both with phrase-based models (which we use) and tree-based models (which require grammatical annotations).

We run Moses as a separate process in server mode. The sentences are sent to the Moses server through Apache's XML RPC client¹⁸ sentence by sentence. We have to send the sentences to Moses server already tokenized and it produces better results if the first letters of sentences are lowercased.

Because of limited computing resources, we trained only English to Czech machine translation, since we consider this translation direction to be the primary one. If we wanted to train Czech to English too, resources required would double. More information about training the machine translation and getting it to work fast can be found in Chapter 6.

The tokenization of the input sentences, however, turned out to be problematic. Because we did not expect any issues, we simply tokenized the input with the OpenNLP tokenizers and we let Moses use its own tokenizers.

Moses, however, tokenizes English slightly differently. This causes problems with the contractions of English suffixes like -n't and -'ll. For example, we tokenize don't as do n't, Moses tokenizes it as don 't. Although there is only a small number of such cases, the contractions are very widely used and their wrong tokenization dramatically decreases the quality of the translations produced.

For that reason, we had to add several corrections of tokenization for the contraction suffixes. Those corrections are hardcoded in the `MosesSearcher.scala`. An optimal solution would be to use exactly the same tokenizers both in our application and in Moses training; we leave this as a possible future improvement.

5.4 Candidate Ranking

After retrieving candidates for a query, the candidate translation pairs must be ranked according to their quality and their similarity to the queried chunk. We use different methods for ranking the candidates retrieved by the various candidate retrieval methods. In the following section, we will briefly describe the ranking methods for each retrieval method.

All ranking methods are based on a combination of scores. Depending on the retrieval method, we assign every translation pair candidate a set of scores, which are then combined into a final score. Translation pairs are ordered by this score.

To learn the best combination of scores, we manually selected translations for heldout subtitle files and then used those annotations as positive and negative examples for training a model using the WEKA machine learning toolkit.¹⁹

¹⁷<http://www.statmt.org/moses/>

¹⁸It is modeled after `SampleClient` in Moses, <https://github.com/moses-smt/mosesdecoder/blob/master/contrib/server/SampleClient.java>

¹⁹<http://www.cs.waikato.ac.nz/ml/weka/>

5.4.1 Training the Models

For the manual annotation required to estimate weights for the scores, we started the translation memory with only the corresponding backoff level (e.g. exact matching or fuzzy matching) and manually translated several heldout subtitle files using the translation workspace (no post-editing was done, the annotators were instructed to only choose the best translation suggestion if possible). After uploading and finishing the subtitle files, the class `cz.filmtit.dataimport.training.GenerateTrainingset` can be used to generate a CSV file in WEKA-compatible format which can then be used to train a model.

For creating positive and negative instances, chunks without any suggestions from the translation memory are ignored. If a translation pair was selected, it is used as a positive example. If no translation pair was selected by the user, a random translation pair from the first ten suggestions is selected and used as a negative example.

5.4.2 Ranking for Exact Matches

The ranking for exact matches uses a linear regression model with the following scores:

- **translation pair count**

The translation pair count specifies how often a translation pair has been observed. This score is calculated as $\frac{c(p)}{\sum_{p' \in P} c(p')}$, where $c(p)$ is the count for the translation pair p and P is the set of all translation pair candidates for a query.

- **source-chunk edit distance**

The edit distance between the queried chunk and the source chunk of the translation pair candidate; the score is normalized by the length of the queried chunk.

- **genre match**

Percentage indicating how much the genres of the queried media source overlap with the genres of the sources of the translation pair.

- **length difference between source and translation**

The length difference between the source and target side of the translation pair candidate may indicate that an alignment is not of a high quality – in most cases of well-aligned texts, the source and target side does not differ in length significantly.

- **final punctuation**

Feature indicating whether the final punctuation of the queried chunk and the candidate translation match.

5.4.3 Ranking for Full-Text Search Matches

- **full-text search score for the document**

The score assigned by the Postgres full text search for the document that represents the translation pair candidate.

- **translation pair count**

The translation pair count specifies how often a translation pair has been observed. This score is calculated as $\frac{c(p)}{\sum_{p' \in P} c(p')}$, where $c(p)$ is the count for the translation pair p and P is the set of all translation pair candidates for a query.

- **source-chunk edit distance**

The edit distance between the queried chunk and the source chunk of the translation pair candidate; the score is normalized by the length of the queried chunk.

- **genre match**

Percentage indicating how much the genres of the queried media source overlap with the genres of the sources of the translation pair.

- **length difference between query and source**

The length difference between the queried chunk and the source side of the translation pair candidate

- **length difference between source and translation**

The length difference between the source and target side of the translation pair candidate may indicate that an alignment is not of a high quality – in most cases of well-aligned texts, the source and target side does not differ in length significantly.

- **final punctuation**

Feature indicating whether the final punctuation of the queried chunk and the candidate translation match.

5.5 Merging Similar Candidates

After all translation pair candidates are retrieved and ranked on each level, we attempt to merge translation pairs that are too similar, not to present the user with several choices that differ only very slightly.

As a translation pair merger, we currently use an implementation based on Levenshtein distance, which removes all candidates whose similarity to a higher-scoring candidate are below a given threshold (1 by default).

5.6 Technical Issues

5.6.1 Concurrency

One important issue in the Core Translation Memory is that the service must be able to handle a large number of simultaneous requests without blocking or long delays. There are, however, some searchers, that are inherently not thread-safe.

To approach this issue, we used the Scala akka library,²⁰ which provides methods for parallelization based on the *actor model*. A `TranslationPairSearcherWrapper`²¹ is a `TranslationPairSearcher` that is constructed with a number of other `TranslationPairSearchers`, which act as the wrapper's workers. When the `TranslationPairSearcherWrapper` receives a request, it routes it to a free worker or adds it to the queue if none of the workers are free. Using this wrapper class, any non-thread-safe subclass of `TranslationPairSearcher` can easily be queried in parallel.

We use the same method for parallelizing the tokenization of `Chunks`, since it is not thread-safe either.

5.6.2 Keeping the Database Up-To-Date

Since we want to ensure good retrieval performance, the core database is read-only in production mode. The user's translations that are stored in the User Space are periodically added to the core database as new translation pairs and the indexes of all searchers are updated in turn.

²⁰<http://akka.io/>

²¹See the class `cz.filmtit.core.concurrency.searcher.TranslationPairSearcherWrapper`.

Chapter 6

Training Machine Translation

Although our initial plan was to show only the translation memory to the translator, we found out that we can also use the corpus to train a model for statistical machine translation. However, we also found that despite getting quite good results from the translation, the durations for getting the translations were too long, compared to the translation memory.

In this chapter, we will describe, how we trained the models, our results and how we managed to get the time cost down. The final model is a part of our project; the script to produce the model from the aligned data is not included, since it would require changes to EMS source code¹, which are untested at the moment and beyond the scope of this project.

Since we focus mainly on English to Czech translation, we modeled, trained and tested only for this way of translation. Training for the opposite way of translation would be possible, too, but all the necessary resources would double, while it would only benefit a small number of people.

This chapter requires some knowledge of phrase-based translation models and Moses in general. If you are not familiar with either, the short version of this chapter is that we are getting – on subtitles – better results with our machine translation than Google Translate, we are getting it fast, and we were able to fit it on a rather small virtual machine.

6.1 Initial Setup

6.1.1 Initial Settings

For training machine translation, we use our subtitle corpus, aligned by techniques described in Section 4.5. Since we want to be more exact in this case, we used a more strict alignment, with 0.6 s tolerance.

We do not require all the metadata about movie files for this task, so we added all the data together. We took away 15,000 sentences for MERT tuning (which proved to be too much) and 5,000 sentences for testing. The rest is training data.

¹EMS is the Experiment Management System, a system for managing Moses experiments, <http://www.statmt.org/moses/?n=FactoredTraining.EMS>

Because of the way we took the testing sentences, they are all from different movies than the training data, except for at most one movie, that can have different sentences both in tuning and testing.

We run the standard EMS² scenario on our data, which includes testing and counting BLEU score³. Although EMS counts both case-sensitive and case-insensitive BLEU, we chose the case-insensitive one.

We chose EMS instead of UFAL’s own *eman*, simply for better documentation of the former and the inclusion of EMS in the standard Moses installation.

Initially, for the language model, we chose the IRSTLM⁴ language model. For the translation model training, we used standard GIZA++.

We also measured the time for running the test translation.⁵ Time was measured on a machine with Intel Core2 Quad CPU Q9550, with 4 cores with 2.83GHz, and 4GB of memory.

For reference on quality, we also translated the test set with Google Translate.

We can see the initial set-up in Figure 6.1.

6.1.2 Initial Results

The initial results are shown below.

Type	BLEU	incomplete average tps
Initial setup	23.88	147 ms
Google Translate	19.07	N/A

Table 6.1: Initial results (note that the time is not directly comparable with the times below due to the pre-filtering, as described below)

We were very pleasantly surprised that our initial results were better than Google Translate, without really doing anything “smart” on our part. This is, however, definitely caused by a smaller domain. On the *newstest2012* set, Google’s results were still quite good, while ours were terribly low; the phrases in newspapers are completely different from phrases in subtitles.

The time looked good on the first glance; however, real times observed did not line up with the results of using the model. The difference was caused by EMT filtering (seen in Table 6.1 as “filter”). EMT, before doing the final tests, actually filters the models in such a way that only needed parts are loaded; the actual time of the final test is, then, much lower.

Also, even when we did not optimize for time spent on initial learning, the learning phase was too long to make any experiments effective. We addressed this issue, too.

²EMS is the Experiment Management System, a system for managing Moses experiments, <http://www.statmt.org/moses/?n=FactoredTraining.EMS>

³BLEU is an automatic metric for measuring the success of a translation

⁴<http://hlt.fbk.eu/en/irstlm>

⁵This was achieved by adding time tracking code to the EMS source code; this code was submitted to Moses and actually “survived” in the main git repository for a few weeks, but was ultimately deleted, because it broke some edge cases.

6.2 Changes in the Setup

6.2.1 Tuning Set Size

The biggest issue with the training was actually the MERT tuning. If we cut down the number of phrases for MERT tuning, the time for tuning decreases significantly, while the BLEU doesn't decrease that much.

Tuning size	BLEU	Tuning time
15,000	23.88	12 hours 57 minutes
1,000	22.95	16 minutes

Table 6.2: Tuning set size decrease

6.2.2 Turning Off the Filtering

To know the real time of the translation, we needed to turn off the filtering in EMT, which required some slight modification in the source code.

However, after that, we found out, that unfiltered models simply will not fit into the memory of the machine. Therefore, we had to both binarize the models and prune the phrase table.

6.2.3 Pruning the Phrase Table

Retrieving suitable elements from the translation table takes most of the time of the entire translation task. Therefore, making it smaller would make the translation task quicker.

As a first idea, we tried to prune the phrase table by using some factoring like short stems instead of full words. However, this showed up as a dead end; too short stems produce an absolutely empty phrase table, and longer stems do not shrink the phrase table significantly.

Instead, we tried another method, described in one of the papers⁶. It prunes the table using statistical significance test, called Fisher's exact test⁷.

This method is implemented in Moses in `sigtest-filter` script; however, EMS doesn't support it in the main Moses repo. We had to add support for it to EMS, only to find out afterwards that Aleš Tamchyna from ÚFAL has already did that.

As described in the original paper, pruning the table actually did increase the BLEU score; what we lost by decreasing the tuning size, we got back by doing the pruning.

⁶H. Johnson, J. Martin, G. Foster and R. Kuhn. (2007) *Improving Translation Quality by Discarding Most of the Phrasetable*. In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pp. 967-975.

⁷http://en.wikipedia.org/wiki/Fisher's_exact_test

Pruning	BLEU	ttable size (gunzipped)
No	22.59	2.10 GB
Yes	23.26	598MB

Table 6.3: Pruning the phrase table

6.2.4 Binarizing the Models

All models – reordering model, language model and translation table (both pruned and upruned) – can be binarized. Binarization is a process in which the table is written in such a format that can be read from disk and loaded to memory only on-demand.

Binarizing is done primarily for a decrease of the memory requirement, not for making the translation faster. Actually, we originally expected the binarized models to be slower, since reading from disk is slower than reading from memory.

Binarizing is also not a part of EMS (probably because it is expected to use the filtering, which cleans the models according to the test set), so we also had to add it to the source.

With the experiments, we *had* to binarize the reordering models and the language models, because the unbinarized versions do not fit into the 4GB memory. However, with the pruned translation table, we could try both binarized and unbinarized version, because unbinarized version fits into memory.

In the tables below, we will write *Time per sentence* as *tps* to save space.

Binarized p.t.	Pruned p.t.	used RAM ⁸	tps with 4 threads	tps with 1 thread
No	No	out of memory	N/A	N/A
No	Yes	2.4G	64 ms	196 ms
Yes	No	1.2G	284 ms	338 ms
Yes	Yes	1.1G	51 ms	193 ms

Table 6.4: Pruning the phrase table

We can notice that with the binarized models, the translation is actually *faster*, if only for a small margin. We think that this is because the on-demand loading actually loads the needed items into memory, so they are not read from disk.

We can also notice that the unpruned binarized table takes about as much memory as the pruned binarized table, but the time per sentence is much bigger.

Another issue is that for the pruned table, the time is almost inversely proportional to the number of threads, but not in the case of the unpruned table. We can only speculate why this is the case; maybe it is because the disk performance is “hitting” us more in the case of the bigger unpruned table.

6.2.5 Smaller Cube Pruning

Cube pruning is a way to search the hypothesis space in phrase-based translations⁹. It adds a number of hypotheses into a stack, which is then pruned; this number can be set up as `cube-pruning-pop-limit`.

It has been implemented in Moses and is default when using EMS, the number is set as 5000. We found out that by decreasing this number, we can make the translation faster with only a minor change in BLEU score.

Cube pruning stack size	tps	BLEU
10	4 ms	22.87
100	5 ms	23.21
250	7 ms	23.23
500	9 ms	23.24
625	10 ms	23.24
750	12 ms	23.25
1000	14 ms	23.25
1250	17 ms	23.25
1500	19 ms	23.25
2000	24 ms	23.26
2500	28 ms	23.25
5000	51 ms	23.26
no cube pruning	66 ms	23.26

Table 6.5: Pruning the phrase table

The BLEU score is, except for very small stack size, changed only slightly, while the time is dramatically reduced by a smaller stack size. For that reason, we chose 250 as our stack size; the BLEU loss is not that significant, while we push down the time to 7 millisecond.

6.2.6 Multithreading

Moses can be run in multithreading mode. Too many threads or too few threads can make the translation slow.

We tried the model, described above, on various numbers of threads. (All with cube pruning size 250).

We can see that the time is optimal, when the number of threads is the same as the number of cores (as we could have already guessed before).

Final model

To reiterate, our final Moses settings:

⁹For more information, see *Hierarchical Phrase-Based Translation*, David Chiang, 2007

Thread count	tps
1	21 ms
2	11 ms
3	8 ms
4	7 ms
5	7 ms
6	8 ms
7	8 ms
8	9 ms

Table 6.6: Pruning the phrase table

- uses all models binarized,
- uses a pruned translation table,
- uses cube pruning with stack sized 250,
- uses the same number of threads as cores.

This model (i.e. the actual model files, the `moses.ini` file and the script to run Moses as a server with correct settings) are both part of the project and on the virtual machine.

6.2.7 Virtual Machine

All experiments were done on a testing machine.

The production machine, which we were provided by the university, had less cores and less memory, so we actually *had* to use all binarized models (since unbinarized translation table does not fit into memory) and, because there are only 2 cores, we use only 2 threads. Also, we have a slight problem with the space, since the reordering model alone takes about 7GB of space, and together with the space of database it barely fits on the virtual machine, so because of Moses, we have to run with only 1.5GB of free space.

However, we think that the inclusion of machine translation is worth it, since the translation memory cannot provide results for all cases, while machine translation always returns something, and thanks to the small domain, the results are quite good and fast.

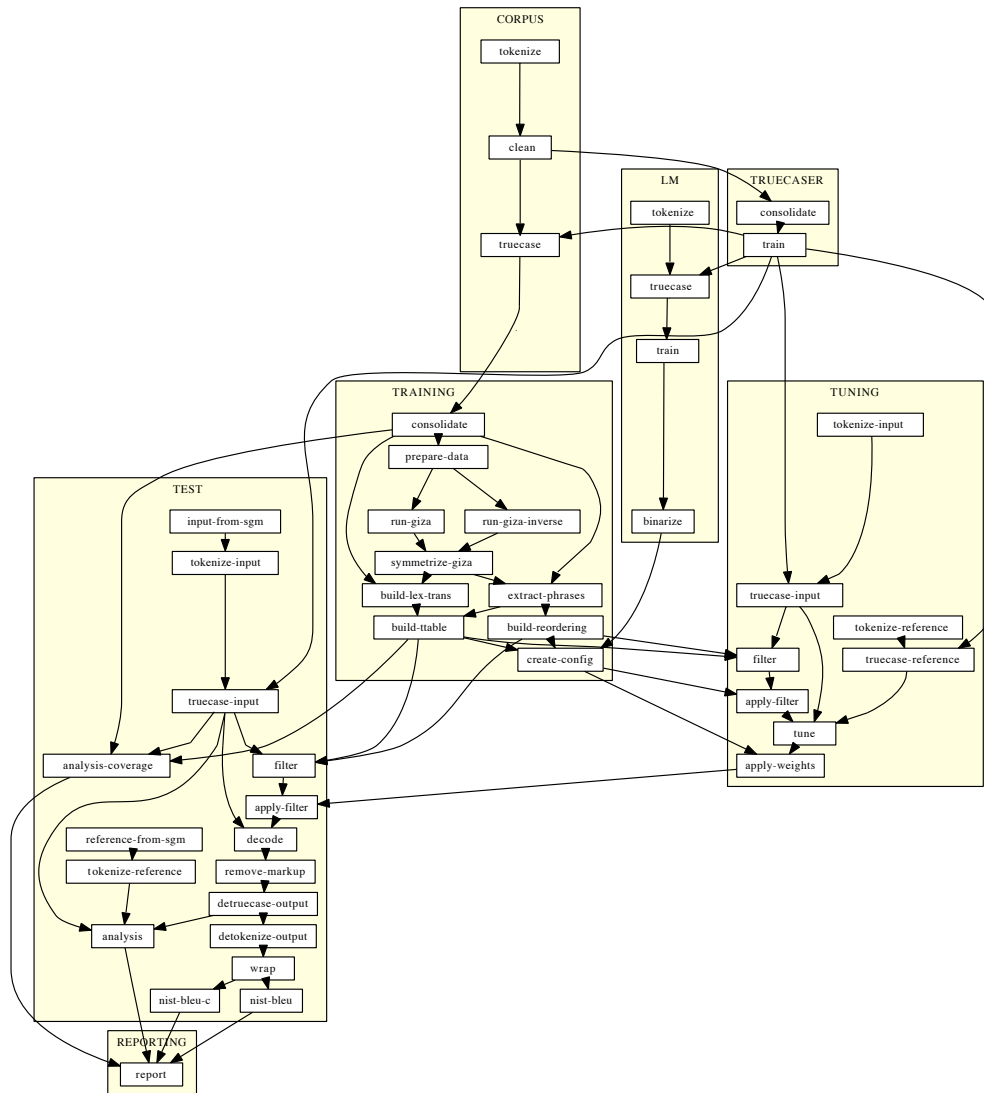


Figure 6.1: Initial EMS set-up

Chapter 7

User Space

7.1 Preliminaries

The User Space component is a server process that provides the web service. Its main tasks are to mediate the services of the translation memory, to reflect all the users' GUI activity and to make the users' work persistent on the server and available always in the state they finished it.

The User Space is a Java Servlet which is run using the *Jetty WebServer*. All of the User Space code is implemented in Java. It uses the *Hibernate* object-relational mapping library. The same *PostgreSQL* database as in the TM Core is used. The in-memory *hsqldb* database is used for unit testing.

Because the TM Core is a separate module, it is also linked as a dependency to the User Space, but they run together as one process in one Java Virtual Machine. Their communication is done by invoking the core methods returning objects of the shared classes.

7.2 Architecture

To make the whole project as clear as possible, we try to use as much as possible from the shared classes and avoid using the User Space specific classes. If some additional functionality is required and cannot be incorporated into the shared classes, mostly the database and core calls, we wrap the shared classes into distinct User Space classes.

The *Server* class which processes the calls on the first level contains the *Session* objects. These objects process the calls with association to the particular logged in users. These two classes are not a part of the set of shared classes.

The session class contains an object representing the user (a wrapper for the shared class). It also contains a hash table of active documents of the user that should be possible to access quickly. Both the document objects and translation result objects (which consist of the source chunk, a list of translation suggestions and the actual user's translation) are wrappers for the shared classes. However, the inner shared objects are used for communication with other components.

7.3 Functionality Overview

The User Space services are available via the RPC calls from the client side. While running the server, there exists one instance of the server class. The main task of the server class is to process the calls from the clients – which in fact means to pass the calls further to the particular sessions (except for calls performed without login) and to manage the sessions themselves.

After a user logs in, a Session object is created. It contains a unique session ID the client uses for authentication in their calls. A session object contains information about the user – the user's settings, a set of the documents owned by the user available via the User object, and a hash table of active documents which are currently in use with loaded chunks. For a logged in user, all of the calls are processed in the session class.

If the user does not explicitly request a document, only basic information about the document remains loaded into memory – document name, movie name, time of last changes of the document, etc. Only when the user opens the document for editing or requests the export of the subtitle file, all the translation results are loaded. However, if the list of owned documents is queried, all of the documents arrive to the client only with the basic information, no matter whether they are already fully loaded in the User Space.

There is also a thread running on the server that checks for how long the sessions have been opened without any user actions. If this time exceeds the predefined session time out limit (either a short limit for common sessions or a long limit for permanent sessions), the session is terminated. Other situations when a session is terminated are when the user logs out explicitly, or in case of a forced log out if the user logs in without having logged out first. When a session ends, basic information about the session is written to the database (user, start time, end time) to enable us to compute statistics about the usage of the application later. The thread also ensures that password change tokens (see Section 8.3.1) and one time authentication identifiers for OpenId (see Section 8.3.2) are also deleted after some time when they are not used.

When a new document is created, the User Space receives basic information about the movie or TV show first. Based on that, it creates a Document object and saves it to the database immediately to receive a unique database ID which is then used as a document identifier in all the following calls. The core is called at this moment to provide a list of possible movies with matching title and year of production together with genre tags obtained from the Freebase knowledge base. The user is then supposed to choose one or none of the suggested movies. This information is used by the User Space at the moment the Core is queried for the translation suggestions.

When a new document is created, the client application sends a list of all chunks that the document consists of, and the User Space adds these chunks to the appropriate document. Then, the client application starts to send batches of chunks to be translated. When the User Space receives such a call, it queries the TM Core for the translation suggestions. Once the suggestions are generated, they are sent to the GUI; they do not persist in the User Space in any way. One reason to possibly keep the suggestions in memory or in the database would be to avoid regenerating them. However, we decided that the cost of regenerating the suggestions is smaller than the cost of additionally storing suggestions in memory or in the database, which would increase the memory requirement significantly. When the client queries saved translation results, they will be sent without translation suggestions. For each untranslated chunk in the document, suggestions are generated and sent to the GUI on demand (see the GUI documentation in Section 9.5.2).

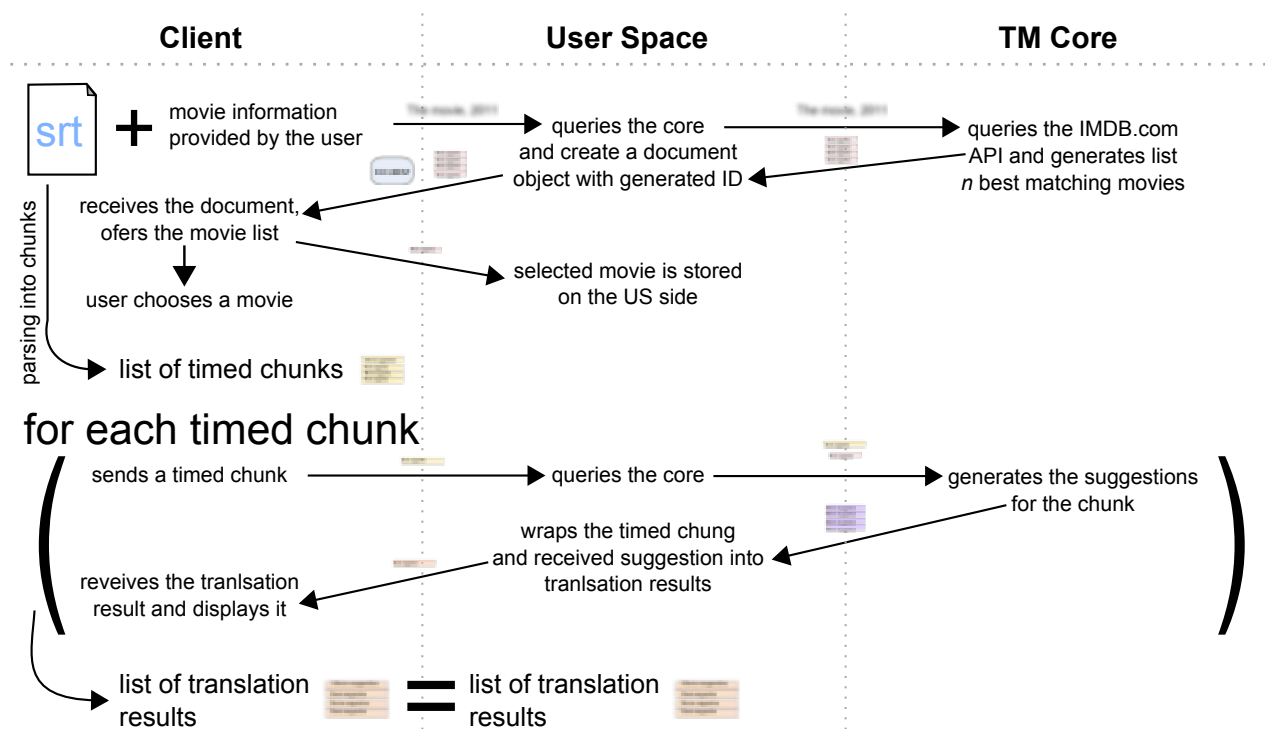


Figure 7.1: Structure of component communication during a new document creation

User Space also provides feedback to the translation memory. A list of new translations the users have produced can be generated together with information on which translation pair they used for post-editing. This information is provided to the Data Import module which adds the new translations to the translation memory and also uses the data for training the scoring of the suggestions (see Section 5.4 in the Core documentation).

Export of the subtitle files is also solved in the User Space. It is described in more detail in Section 7.6.

7.4 User Management

Because our own experience was that requiring a registration from a web service often prevented us from using it, we decided to make the registration as easy as possible. We have no constraints on user names (except for non-emptiness) and very little constraints on password (only it must be at least 3 characters long).

We leave the assessment of the strength to the user. The password strength constraints are often rather arbitrary and many classes of strong passwords do not pass them, it is hard to develop a reasonable strength measure.

We allow the user to work in multiple browser tabs or windows, provided it is the same session (i.e. it has the same `SessionID`), because we believe it is natural to work on multiple translations at once, to have the Document List opened to see the translation progress, etc. The

user himself is responsible for having the same document opened multiple times, which is not expected behavior and so changes to the document in one Translation Workspace are not projected into the other Workspace until reloaded.

However, we decided to disallow one user having multiple sessions at one time, since it would cause problems to us when implementing it and it doesn't happen very often in our opinion. We do not support multiple users collaborating on one subtitle file at the moment.

As described in 7.5.4, if a user registers through OpenID, we have to assign a user name to him (the unique OpenID identifier we get is typically a long hash, which does not look like a nice user name), see Section 8.3.2. Because the user name is at least partly generated, the user might not like it. Therefore, we offer each user the possibility to change the user name (provided that the one he chooses is free). The user name is only used to authenticate the user (if not using OpenID login) and is displayed in the top right corner in the application; the internal unique identifier of a user is his user id, an integer assigned to each user on registration that never changes.

7.5 Implementation Details

7.5.1 Data Types Overview

The User Space uses the shared classes used throughout the application, or wrappers of such classes. A brief overview of the classes used and their roles in the User Space follows. We use the abbreviation US to refer to the User Space in the following sections.

- **FilmtitBackendServer** – The class is the Java Servlet whose public methods are invoked by the RPC calls. Its main task is to mediate the calls to concrete opened sessions and take care of users login. A thread checking if there is a session without activity for a long time and terminating the non-active sessions is run in the server class.
- **Session** – The class represents a running session and processes the client calls. There is exactly one session object for one logged in user. It contains hash tables of documents being currently edited to make them quickly accessible based on their IDs. During the existence of a session object, all the client-side operations are stored in memory and saved immediately to the database. When the session ends – either by the user logging out or by exceeding the maximum time without a user action – a record about the session is stored to database (ID of the user that owned the session, its start time and end time). This is intended as an activity log from which further statistics can be inferred.
- **USUser** – This class is a wrapper of the shared User class. During the existence of the session, it is used as a provider of the documents owned by the user. It is also used for storing the settings of the user.
- **USDocument** – The class is a wrapper of the shared Document class. A USDocument object represents a subtitle file with additional information about the movie it belongs to. Its content is supposed to be a mirror image of the Document object on the client side. It contains a list of Translation Results representing the actual subtitle chunks if the document

is loaded to be edited. It is not connected to the translation results by the database mapping to make it easy to send a list of the user's documents without the actual content.

- `USTranslationResult` – This class is a wrapper for the shared `TranslationResult` class. The class congregates the original subtitle chunk, its timing, the translations suggested by the translation memory, and the results of the user activity – the users' translations and information about which translation suggestion they chose. It is the class where the actual translation is performed. Although a user can delete the document, all the translation results are kept in the database to provide feedback to the translation memory.
- `Emailer` – A class used for sending email from the application. It is used for confirmation of user registration and also when the user forgets his password and requires sending a new one via email.

There are also two third-party classes included in the User Space source code. It is the `IdGenerator` class which generates session IDs (distributed under the *Apache License, Version 2.0*) and the `BCrypt` class which ensures hashing the passwords before we store them in the database (distributed under *BSD licence*).

The `SubtitleDownloadServlet` class responsible for exporting the subtitle files is described in Section 7.6. Details concerning the user registration, login and using open ID are discussed in more detail in Section 8.3.1 in the GUI chapter. Details about handling the communication with the client are covered in Section 8 of the GUI chapter.

7.5.2 Database Mapping

As mentioned before, the User Space mirrors all the client's operations and makes them persistent on the server side. The persistence is ensured by saving the data to the database. There are a number of sophisticated tools for the JVM to ensure data persistence, e.g. the *Java Persistence API* framework, or some frameworks working on a higher level of abstraction, such as *Spring Data Framework*. Since only very basic database operations are required during the operation of the User Space – loading and saving of the raw Java objects – only an object relational mapping library is used, namely *Hibernate*. The structure of the database being mapped is displayed in Figure 7.2.

The mapping reflects the data properties of classes. If the class is a wrapper of a shared class, the getters and setters of the properties are bound to the wrapped objects properties.

As mentioned before, the mapping of `USDocument` does not include the list of `TranslationResults` the document contains in order to be able to send the documents both with and without the translation results. It would also be possible to use lazy *Hibernate* collection mappings, but it would cause problems at the time the object are serialized.

7.5.3 Providing Feedback to the Translation Memory

In order to be able to provide feedback to the translation memory and take advantage of the translator work for improving the translation memory, the implementation of deleting a docu-

Userspace database

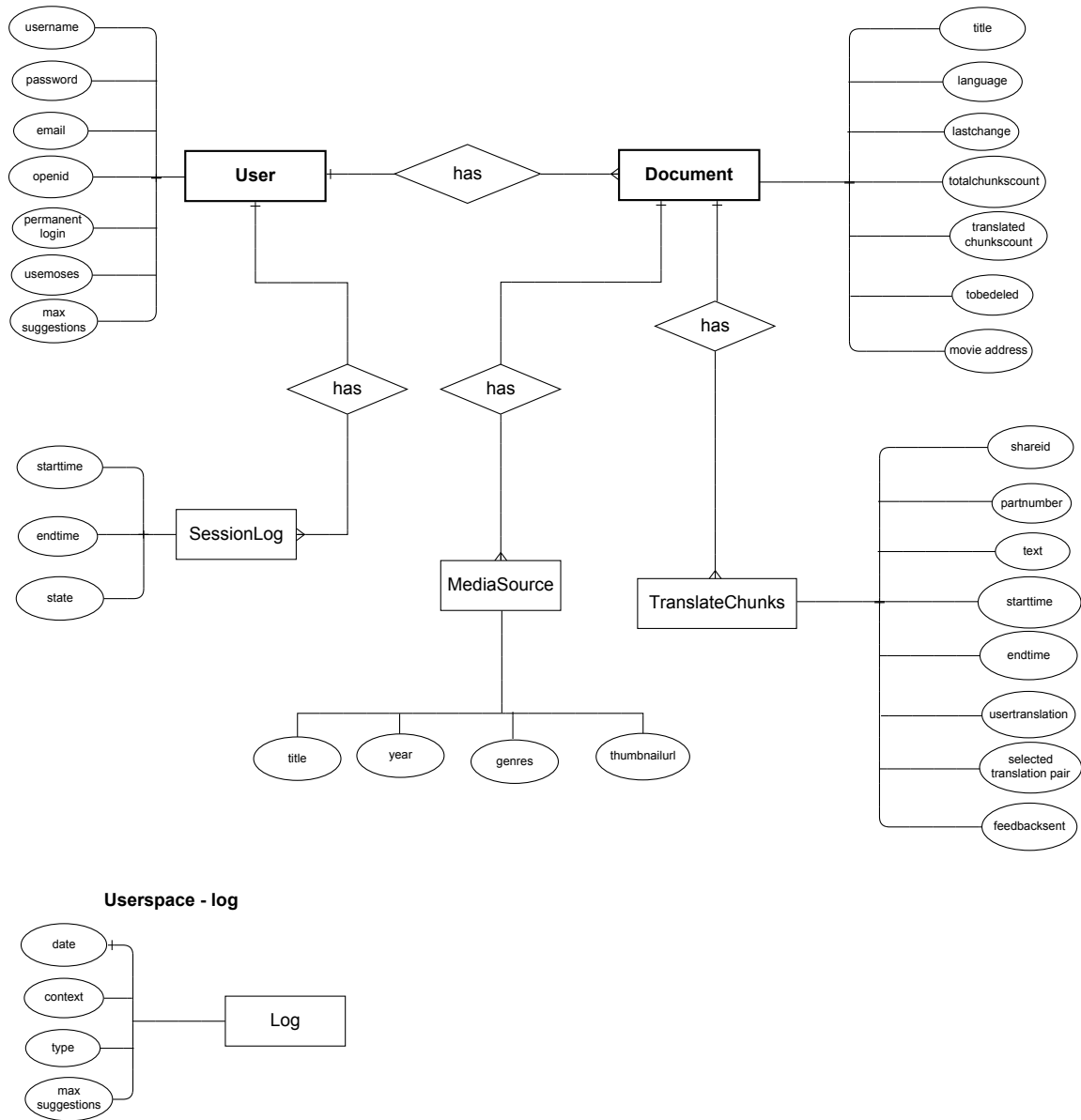


Figure 7.2: EM diagram of the database used by the User Space

ment and the selection of the media source should not be done synchronously. Handling this requirements is described in following paragraphs.

The User Space and the Translation Memory Core use the same database table for media sources (representing the movie or the TV show the subtitles are from). When the users create a document, they are required to enter the title of the movie about to be translated. After that, the *Freebase* service is queried for the movies and TV shows possibly having such title and a list of these matches is provided to the user to choose from. After the user chooses the media source, the media source object is sent back to the User Space. Then, a search in the database table for media sources is performed. If there is a record with the same title and the release year, its data (genres and thumbnails URL) are updated, if it is a completely new media source, it is saved to the table.

The reason for doing this is not to have duplicate media sources in the database and also the fact that the meta data of the movie should be available to the Data Import module when the feedback to the translation memory is provided.

Another issue to be mentioned here is the handling of document deletion. When the user clicks on the delete button in the interface, the document is marked to be deleted in the User Space and is removed from the list of documents owned by the user and from the list of documents that has been loaded during a session. From that time the document is not available to the user. A thread is then run that deals with the translation results the document consists of. All the translation results that have already been used as feedback to the translation memory are deleted from the database. If the document is empty after this step, it is deleted too.

Providing the feedback itself is done by a static method of the `USTRanslationResult` class. This method returns all the translation results what have not been used for feedback yet (hasSentFeedback flag set to false). The translation results are provided with a reference to a document they belong to which allows to resolve the media source later. If the translation results come from a document that has been flagged as ready to be deleted, both the translation results and the document are deleted. Otherwise, the flag informing about having provided the feedback is set.

7.5.4 OpenID

Apart from normal login, we also use OpenID for registration and login.

OpenID¹ is “an open standard that describes how users can be authenticated in a decentralized manner, eliminating the need for services to provide their own ad hoc systems and allowing users to consolidate their digital identities” (from Wikipedia).²

We use *JOpenID* for OpenID authentication. The positive aspect of *JOpenID* is that it is easy to implement and use; the negative aspect is that it does not support arbitrary OpenID URLs, so we (as the web application developers) have to list the OpenID providers that we decide to support and let the user to choose only from these. From *JOpenID* documentation:³

Yes OpenID standards contains two parts: **discovery** and **authentication**. Discovery

¹<http://openid.net/>

²<http://en.wikipedia.org/wiki/OpenID>

³<http://code.google.com/p/jopenid/wiki/FAQ>

is the process where the RP uses email or URL to look up the address of OP. It is complex, and may confuse the end users.

Instead, Web UI designer should give end users the chooses of their familiar OpenID accounts, such as Google account, Yahoo account, etc.

JOpenID only supports OpenID authentication because the discovery protocol is hard to use, and difficult to implement. You will find that JOpenID meets your 99.9% requirements.

We support Google, Yahoo and Seznam.cz;⁴ this is done by implementing `cz.filmtit.userspace.login.SeznamData`. We think that just a very small percentage of users actually know what OpenID is; therefore, allowing to login via Google, Yahoo and Seznam accounts seems sufficient.

However, there is a question to solve – what to do when the user logs in through OpenID for the first time, because the user still needs to be in our database as a regular user. We were thinking about requiring an explicit registration, but that is unnecessary and complicated. Therefore, we register the OpenID user transparently.

Because we register the users as full users, we generate a username and password for them and then send these to their e-mail addresses, which can be acquired from the OpenID provider. The users can thus log in even without their OpenID; they also have the possibility to change their username and/or password on the Settings page.

7.6 Exporting Subtitle Files

Users can export the files in `.srt` format. We decided to ignore the `.sub` format because, as is shown in 4.2, in the initial data, the percentage of usage for `.sub` was minimal and the work needed would be too big.

Exporting the files is a straightforward task – we traverse the chunks, connect them to subtitle items if they have the same time, and save their contents.

The issue with exporting is that we want to make the `.srt` file downloadable. This cannot be done using only GWT and RPC, since GWT works through Javascript, which cannot produce files for download. So instead, we run a separate servlet, whose only responsibility is to produce `.srt` files. This servlet, named `SubtitleDownloadServlet`, is created at the start of the program, has a `FilmTitBackendServer` reference, and receives document ID, session ID and the file type through a standard GET request. First, it is checked if the session ID is correct, then if the users own the given document, and if they do, a file is created. Sequence diagram of downloading the exported subtitles is in Figure 7.3

We can export three different types of file:

- a file with only the translated chunks, completely omitting the untranslated ones,
- a file with the translated chunks, with the source chunks at places where nothing is translated,

⁴<http://seznam.cz>, a Czech web portal, one of the largest Czech OpenID providers

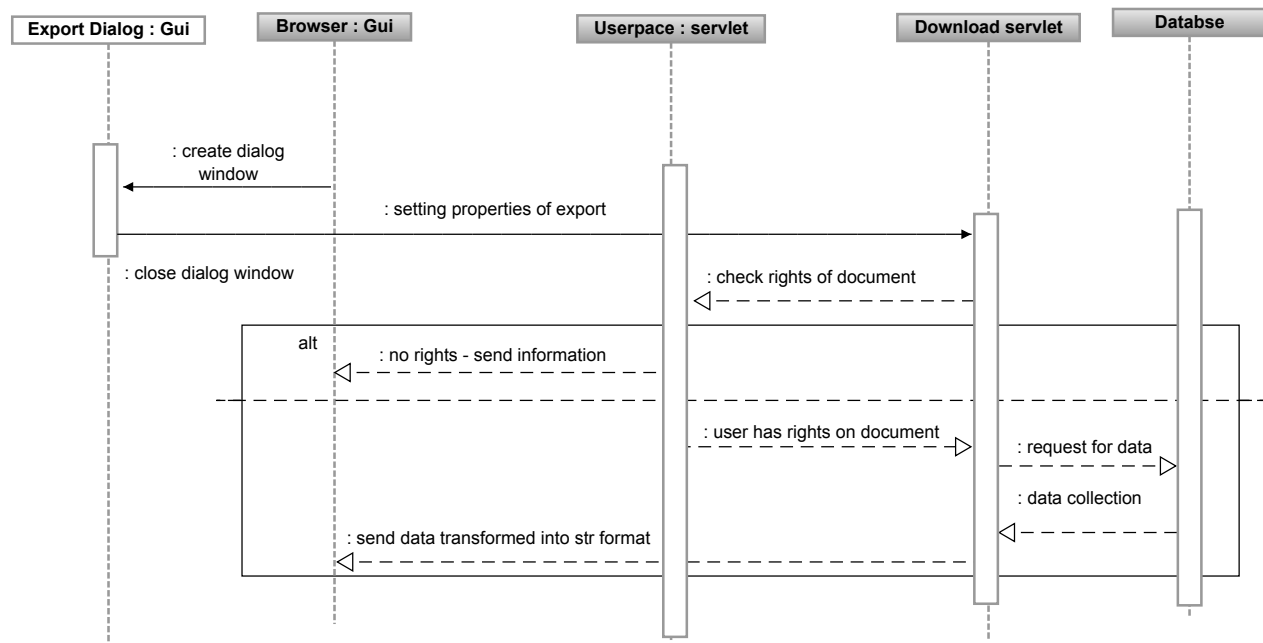


Figure 7.3: Sequence diagram of exporting a subtitle file

- a file with only the source chunks.

All three types are either in .srt or .txt format. The .txt format is just the text of the subtitles without any time information.

Chapter 8

Communication between GUI and US

This chapter describes the communication interface between the GUI and the User Space, which is based on Remote Procedure Calls.

8.1 Remote Procedure Calls

GUI communicates with User Space via the Google Web Toolkit RemoteService implementation.

8.1.1 RPC in general

A *remote procedure call* is a method providing interaction between the client (the GUI) and the server (the User Space).

From the start, it was obvious to us that we want to use a well-established method to implement the Remote Procedure Calls. GWT provides a ready-to-use Remote Service implementation, where the programmer only has to define the set of RPCs as an interface, implement the interface on the server side, and invoke an RPC on the client side by implementing the GWT AsyncCallback interface. Everything else is handled automatically and transparently by GWT.

The type of an RPC parameter or its return value can be any GWT-compilable type. Because this type must be known both to the server and to the client, we use only primitive types and shared classes. Each class sendable through the GWT RPC mechanism must implement the GWT IsSerializable interface (which is only a marker, i.e. it does not have any abstract methods) and a parameter-less constructor; this also holds for the checked exceptions, which are technically only a special type of return value.

8.1.2 Our implementation

The shared class FilmTitService *defines* the FilmTitService interface, an API for communication, by specifying the RPCs as methods with a name, a set of parameters, a return type and checked exceptions. The User Space class FilmTitBackendServer must *implement* each of the defined RPCs. The interface provides *asynchronous* RPC methods – the responses are processed by *callbacks*. A class in `cz.filmtit.client.callables` may *invoke* the RPC, passing the required

parameters to the GWT RPC mechanism which ensures their delivery to the server and the execution of the implementation method.

The callable class must implement an asynchronous callback by implementing the methods `onSuccess()` and `onFailure()`, which are called by the GWT RPC mechanism when the RPC *returns*, i.e. returns a value or fails with an exception and the result is received by the GUI. Failure is defined as the RPC server implementation throwing an exception, which is passed as a parameter to `onFailure()`. Otherwise, the returned value is passed to `onSuccess()`. In case of common points of failure that do not qualify as an exceptional state, such as user registration attempt with an already existing username, the common failure is typically signaled by the return value instead (typically a null or a false).

The RPC is always invoked from the client, because JavaScript security policies do not allow to handle incoming calls. Therefore, in case the User Space needs to actively contact the GUI without the GUI having sent a request before, this has to be implemented in the GUI by polling.

Most of the RPCs can only be invoked by a logged-in user. Such RPCs always have the `sessionId` as their first parameter to authenticate the user and can throw an `InvalidSessionIdException`. A `sessionId` is a unique identifier of a logged in user, valid for 1 hour (or 1 month if the user decided to be “permanently logged in”), managed by the User Space.

The parameters, return values and exceptions thrown must be GWT-serializable, and their types must be supported both by GUI and User Space. Therefore, we either use types supported both by GWT and Java (such as `int`, `String` or `List`) or the Shared classes (see Section 3.2).

8.1.3 Some issues

Because of JavaScript and browser limitations, the RPC mechanism of GWT only allows asynchronous calls from client to server. This suited our needs often, but we had to go around these limits in some situations. Quite often, we need the RPC to be synchronous and blocking, e.g. in logging in or changing user settings. In such situations, the page or dialog is “deactivated” on invoking the RPC (all active components are disabled – e.g. the text boxes are read-only, the buttons do not respond to clicks), a reference to the page or dialog is passed to the class invoking the RPC, and when the RPC returns, the page or dialog is “reactivated”, either with a success message or an error message, depending on the result of the RPC.

In one case, we would need an opposite direction RPC, i.e. the server to invoke the RPC on the client. This is when authenticating the user through OpenID, where we have to perform the authentication in a new window – the new window then receives the result of the authentication, but it cannot be easily passed directly into the opener window. Here, we decided to use active polling – the opener window regularly invokes the `GetSessionID` RPC, which returns null if the authentication process has not yet finished, this being a signal for the GUI to continue polling. To link together the opener window and the authentication window, a unique `authID`, generated by the User Space, is used.

Failed RPC Requests Handling

We decided that our general error handling strategy should be to try to solve all problems without user interaction, and to inform the user only about user-generated errors (such as incorrect SRT

file format or session timeout) or critical errors (such as permanent loss of connection to server).

The first thing was to turn the calls into objects, so that they store their parameters and can be called again on failure. We also created a generic superclass for them, `Callable`, filled with the necessary boilerplate code and methods performing the default actions, to be overridden if necessary.

We identified the response that we usually get in case of network problems (it is a `StatusCodeException` with the status code 0). The problems can be temporary or permanent; in case of temporary problems, the default response of showing an error message was found to be highly inappropriate, as it would suffice only to resend the call. Therefore, we introduced retrying the call for three more times in such a case with short delays, before showing an error message to the user.

The retrial of calls has proven to be an efficient way to deal with problems, so we decided to extend it to all types of failures. However, in case of non-network errors, we soon found that retrying the calls does not make sense in some situations; so we decided to keep the retrial as a default action which can be overridden in the extending classes.

We also added a timeout for each request after which the request is regarded as lost and is retried. Such a situation is not at all typical, but we need to expect it to prevent the user from “starvation” (the user can always escape from any state by refreshing the page in the browser, but this can lead to an uncontrolled local data loss or even to objects being left in an inconsistent state – such as a document without content).

8.2 Manipulating Documents

A document is always identified by its `documentID` (a long number generated by User Space on document creation), therefore most of the methods have the `documentID` as one of their parameters.

A chunk in the document can always be identified by its `ChunkIndex`, but for convenience in some cases the whole chunk is sent instead. First, we introduce all the RPCs manipulating the documents. Then, we illustrate a typical sequence of RPCs invoked on document creation.

8.2.1 Sending translation results – discussion

At first, the translation suggestions from the Translation Memory were requested and sent for each chunk individually. This had the theoretical advantage that the user received each of the translation suggestions as soon as possible. However, we found the overhead of the RPC calls to be unreasonable. Not only was the overall time necessary to load all the suggestions too long, but also a typical browser was unable to handle such a large number of requests and usually froze for long periods of time, often unfreezing only after all of the suggestions arrived.

A simple solution seemed to be sending all of the chunks for translation in one request. However, such a request can take several minutes to complete, and we still wanted to keep the nice property of the application of showing the suggestions as soon as possible at least for the first lines – so that the user can start the translation immediately. Thus, we decided to send

the request in batches, starting with a small size of the batch for the beginning of the file and gradually increasing the batch size.

The first way to do that was an exponential increase, where each batch contains twice as many chunks as the previous one, starting with a batch of size one. This proved to solve the aforementioned problem of browser freezing; however, another problem arose. As the batch sizes grew larger and larger, the later requests often took several tens of seconds or even minutes to complete. Meanwhile, the user would often navigate away from the Translation Workspace, rendering the translation suggestions, including the currently being processed ones, obsolete. This was of course detected in GUI and the suggestions were thrown away on arrival, but still it meant that the server was working in vain for some time, possibly resulting in an unnecessarily slower responsiveness to other users (and even to the very same user as well).

We introduced two further solutions. The first and simple one was limiting the size of the batch to 64 chunks (which typically take at most 5 seconds on the server side), so that the amount of useless work could not grow too large. The second solution was to stop the translation suggestions from being generated on the server, by issuing a `StopTranslationResults` request. (This breaks the typical model of processing the queries, where once a query is sent, its processing is not influenced by queries that arrive later.)

The two major issues with the `StopTranslationResults` request were:

- to interrupt the suggestions generating which is already in process (which is done in parallel for the chunks in the batch, making this even harder).
- to identify what to stop and what to let continue.

The first idea was to mark the whole document as closed, and to check this when generating the suggestions. However, this would cause problems if the same document was immediately reopened by the user, and even larger problems if the user already had the document opened twice – this can easily happen by mistake, the logical reaction of the user is then to close one of the Translation Workspaces, and then he would stop receiving suggestions even in the other opened Workspace. Thus, we then decided to create an identifier for each Translation Workspace instance and to bind the `StopTranslationResults` request to this identifier. However, we soon found out that this would require additional overhead only to handle this probably not very common situation, so we dropped that idea as well.

Finally, we decided for a compromise solution between those two. The `StopTranslationResults` request is now bound to individual chunks, which now have an added boolean field `isActive`. This field is set to true when the `GetTranslationResults` request arrives to the server, for each of the chunks on the request. The `StopTranslationResults` request contains a list of the chunks which translations should be stopped. On arrival of that request, User Space simply sets the `isActive` field to false – because User Space and Core share the same memory and because only references to the chunks are passed around, this change is visible in core even though the chunks have already been sent for translation. The necessary overhead is minimal, and there is no "false alarm" in the typical case. The only unsolved case is the situation where the same batch of chunks are sent for translation at the same time from two different instances of Translation Workspace with the same document opened, and then one of these Workspaces is closed, causing the suggestions generation to stop even for the batch from

the other Workspace. Although this situation is unlikely and non-critical (the suggestions would be missing only for this one batch, the later batches would be translated correctly), we still offer a general solution by repeating all request on failure (unless it is obvious that the failure will reoccur).

8.2.2 Whole Document Operations

`DocumentResponse createNewDocument(String sessionId, String documentTitle, String movieTitle, String language, String moviePath)`

Creates the document (without source chunks, which have to be added by calling `saveSourceChunks`). Returns the `documentID`, together with media source suggestions based on `movieTitle`.

`Void selectSource(String sessionId, long documentID, MediaSource selectedMediaSource)`

Sets the media source of the document. The media source represents the movie or series which the subtitles come from.

`List<Document> getListOfDocuments(String sessionId)`

Returns all documents owned by the user, ordered by date and time of last change.

`Document loadDocument(String sessionId, long documentID)`

Returns the document with the given `documentID`, with source chunks but without translation suggestions (these have to be explicitly requested by `getTranslationResults`).

`Void changeDocumentTitle(String sessionId, long documentID, String newTitle)`

Sets a different title for the document.

`List<MediaSource> changeMovieTitle(String sessionId, long documentID, String newMovieTitle)`

Returns media source suggestions based on `newMovieTitle`. The movie title is not changed yet: this is only done on calling `selectSource`.

`Void deleteDocument(String sessionId, long documentID)`

Remove the given document from the list of user's documents. (The data might not be discarded immediately as the translations still might be used to enrich the translation memory)

8.2.3 Source Subtitles Operations

`Void saveSourceChunks(String sessionId, List<TimedChunk> chunks)`

Save the given source chunks as the contents of the given document (which was already created by calling `createNewDocument`).

`Void setChunkStartTime(String sessionId, ChunkIndex chunkIndex, long documentID, String newStartTime)`

Change the start time of the given chunk to the new value. The value must be in the SRT format, i.e. hh:mm:ss,ttt.

`Void setChunkEndTime(String sessionId, ChunkIndex chunkIndex, long documentID, String newEndTime)`

Change the end time of the given chunk to the new value. The value must be in the SRT format, i.e. hh:mm:ss,ttt.

`Void setChunkTimes(String sessionId, ChunkIndex chunkIndex, long documentID, String newStartTime, String newEndTime)`

Change the start time and end time of the given chunk to the values. The values must be in the SRT format, i.e. hh:mm:ss,ttt.

`TranslationResult changeText(String sessionId, TimedChunk chunk, String newDbForm)`

Change the source text of the chunk, resulting in new translation suggestions which are sent as the result.

`Void deleteChunk(String sessionId, ChunkIndex chunkIndex, long documentID)`

Remove the chunk from the document, together with its translation if it exists.

8.2.4 Target Subtitles Operations

`TranslationResult getTranslationResults(String sessionId, TimedChunk chunk)`

Get the list of possible translations of the given chunk. The `TranslationResult` instance contains zero or more translation suggestions, which come from the Translation Memory and/or the Machine Translation System.

List<TranslationResult> getTranslationResults(String sessionId, List<TimedChunk> chunks)

Get the list of lists of possible translations of the given chunks. Each TranslationResult instance contains zero or more translation suggestions, which come from the Translation Memory and/or the Machine Translation System.

Void stopTranslationResults(String sessionId, List<TimedChunk> chunks)

Stop generating translation results for the given chunks (to be called after getTranslationResults has been called with the given chunks but the results are suddenly not needed anymore).

Void setUserTranslation(String sessionId, ChunkIndex chunkIndex, long documentID, String userTranslation, long chosenTranslationPairID)

Save the user translation of the given chunk (no matter whether it is user's own translation or a suggestion taken over or post-edited). The ID of the TranslationPair chosen for post-editing is also sent, providing feedback which then can be used to improve future suggestions.

8.2.5 Document Creation

The sequence diagram 8.1 illustrates a typical sequence of RPCs invoked to create a document, including interaction with the Freebase service and the Core.

First, an empty document is created by the createNewDocument RPC. The Freebase service is queried for suggestions on the actual movie or series matching the given movieTitle, from which one is then selected by the selectSource RPC – this enables the Core to provide translation suggestions ranked according to the closeness of movie genres. Meanwhile, the saveSourceChunks RPC is invoked to store the contents of the document once the subtitles file is parsed.

When both selectSource and saveSourceChunks have returned, the getTranslationResults RPC starts to be iteratively invoked to obtain translation suggestions, generated by the Core, until the whole document gets translation suggestions.

8.3 User Registration and Login

The user is required to log in to use the application.

The users are logged in if they have a valid sessionId which is linked to their user accounts. The sessionId expires after a given period of time without any user interaction with the server (1 hour by default).

Two ways to log in are supported – Simple Login and OpenID Login; these are further described separately. The only two common methods are checkSessionID and logout, which are therefore described first.

SessionResponse checkSessionID(String sessionID)

Validates the given sessionID. To be used with a sessionID that does not result from invoking neither simpleLogin nor getSessionID (such as a sessionID stored in a cookie). Returns a SessionResponse instance containing the sessionID and the User object if the sessionID is valid, or null otherwise.

Void logout(String sessionID)

Invalidate the user session with the given sessionID.

8.3.1 Simple Login and Registration

The Simple Login is the classical implementation of user login. Each user must first register with a unique username and a password of choice. These credentials are then used to log into the application. (The password is to be stored on the server side in the form of a one-way hash.)

The users can also provide an e-mail address for forgotten password retrieval. They then has the option to request a password reset e-mail, based on his username or the e-mail address. The password reset e-mail contains a link to a page where the user can set a new password, based on a temporary password change token.

Boolean registration(String username, String password, String email)

Register a user with the given username and password, also setting the e-mail address if provided and sending registration info to it. Returns true on success, false if the username is already taken, and an exception in case of other errors (usually an invalid e-mail address).

SessionResponse simpleLogin(String username, String password)

Try to log in the user with the given username and password. Returns a SessionResponse instance containing the sessionID and the User object on success, or null if the credentials are invalid.

Boolean sendChangePasswordMail(String username)

Send a password reset e-mail to the e-mail address of the user with the given username. Returns true if the e-mail is successfully sent; returns false if the username is not registered or there is no e-mail address stored with the corresponding user account.

Boolean sendChangePasswordMailByMail(String email)

Send a password reset e-mail to the given e-mail address. The password reset link is bound to a username; therefore, if there are multiple user accounts with the given e-mail address, multiple password reset e-mail are sent to the e-mail address. Returns true if the e-mail is successfully sent; returns false if the e-mail address is not registered with any user account.

Boolean changePassword(String username, String password, String token)

Set a new password in case of forgotten password. The temporary password change token must be still valid, and identical to one sent in the password reset e-mail to the user with the given username. Returns true if the password change is successful; returns false if the token is not valid.

8.3.2 Login via OpenID Services

This section describes the OpenID Login process. First, the three RPCs involved are introduced. Then, the whole process, also shown in the sequence diagram 8.2, is described.

LoginSessionResponse getAuthenticationURL(AuthenticationServiceType serviceType)

Return the URL of a window to show to the user to log in using his OpenID account at an OpenID provider specified by `serviceType`. It leads to a web page of the OpenID provider, with the return page set to the FilmTit application, to the `AuthenticationValidationWindow` page.

A generated temporary one-time identifier, `authID`, is also included in the response. It is used to pair the authentication process, which takes place in the newly opened window, with the main GUI window.

Boolean validateAuthentication(int authID, String responseURL)

Validate the response URL from the OpenID provider, which contains information about the result of the OpenID authentication.

If the authentication is found to have been successful, a new session is generated for the user and paired with the given `authID`, and true is returned. Otherwise, the method returns false.

SessionResponse getSessionID(int authID)

Check whether the user has already successfully logged in using his OpenID with the given `authID`.

- If yes, the corresponding `sessionID` and `User` object are returned.
- If not yet, but the `authID` is valid, which means that an OpenID login operation is probably still in progress, null is returned.
- Otherwise, an exception is thrown.

The OpenID login process

The authentication itself is done in a new authentication window. A successful authentication in the authentication window is then paired with the GUI main window using a temporary one-time identifier, `authID`, shared by the User Space, the main GUI window and the authentication window. (Because of JavaScript security restrictions, there is no simple way of sending the result

of the authentication process from the authentication window to the main window; however, the `authID` can be sent to the new window easily because it is already known at the time of its creation.)

When OpenID Login is requested by the user, the GUI main window calls `getAuthenticationURL` method. The User Space generates an `authID`, and a URL to be used for the authentication, and returns that to the GUI, which opens a new authentication window with the received URL. The URL points to an OpenID provider web page (currently supported OpenID providers are Google, Yahoo and Seznam). As a GET parameter, it contains the return URL, which leads back to FilmTit – to the `AuthenticationValidationWindow` page, including the `authID` as a GET parameter in the return URL.

After opening the authentication window, GUI starts waiting for the user to authenticate. The waiting is active, polling the User Space with `getSessionID` in regular intervals until a non-null value is returned.

Meanwhile, the user can authenticate in the authentication window. The OpenID provider then redirects the user to the return URL, which is the `AuthenticationValidationWindow`, together with parameters describing the result of the authentication and providing some information about the user. The whole response URL is then sent through `validateAuthentication` to User Space for validation.

If User Space finds the authentication to be successful, a new session is created for the user, and the `sessionID` is stored in a pair with the `authID`. If a user with the given OpenID is not registered yet, registration is transparently performed at this moment; no explicit registration is required for OpenID login. The authentication window is then closed. In case of an error, the error message is displayed in the authentication window (and the window stays open).

When a successful authentication took place, the `getSessionID` method returns the `sessionID` and a `User` object, the polling stops and the authentication process is completed.

8.4 User Settings

There are several settings that the user can change. There is a separate call for each of the settings – therefore, a set of individual calls must be invoked if the user decides to change multiple settings at once, and each of the calls can succeed or fail independently on the results of the other calls.

The settings are stored in the `User` object and sent to GUI on login, as a part of the `SessionResponse` which is the result of the methods `simpleLogin`, `getSessionID` and `checkSessionID`. There is no dedicated method to load the settings; `checkSessionID` is to be used for that purpose.

8.4.1 Account and Logging in Settings

`Void setUsername(String sessionID, String username)`

Change user's username.

Void setPassword(String sessionID, String password)

Change user's password.

Void setEmail(String sessionID, String email)

Change user's e-mail.

Void setPermanentlyLoggedIn(String sessionID, boolean permanentlyLoggedIn)

Stay logged in permanently (for 1 month) instead of 1 hour (sets the session timeout). The time out times configurable, for details see [11.3.5](#).

8.4.2 Translation Workspace Settings

Void setMaximumNumberOfSuggestions(String sessionID, int number)

Set maximum number of suggestions to show for each line.

Void setUseMoses(String sessionID, boolean useMoses)

Include Machine Translation results in translation suggestions.

8.5 Remote Logging

The remote logging is used to log messages from GUI on server. The messages are to be printed in the server log and can also be stored in the database.

Void logGuiMessage(LevelLogEnum level, String context, String message, String sessionID)

Log the given message from GUI on server. The supported levels are *DebugNotice*, *Notice*, *Warning* and *Error*. The context specifies the type of the message and should be constant for each instance of a similar message; it can contain e.g. a class name, a method name or an RPC name. The message should be as detailed as to provide all necessary information, including e.g. values of variables or a stack trace if applicable.

The sessionID is only used to include the userID in the logged message; it is to be null if the user is not logged in.

8.6 Exceptions Thrown by RPCs

The RPCs typically throw an exception on failure. These exceptions should be of only four types, described in this section.

8.6.1 InvalidSessionIdException

Most of the RPCs can only be invoked by a logged in user – such RPCs take the `sessionId` as their first parameter and throw an `InvalidSessionIdException` in case of an invalid `sessionId`. Typically the ID would be originally valid but expired, so the expected reaction to this exception is to simply ask the user to log in again.

8.6.2 InvalidDocumentIdException

RPCs that manipulate the document usually take the `documentID` as a parameter and throw an `InvalidDocumentIdException` if a document with the given ID does not exist or does not belong to the user.

8.6.3 InvalidChunkIdException

RPCs that manipulate individual chunks take either the whole chunks or only their indexes as a parameter. They throw an `InvalidChunkIdException` if the specified chunk does not exist.

8.6.4 InvalidValueException

The `InvalidValueException` is thrown by a method if a value provided by the user, such as an e-mail address or a password, does not have the required format. It always contains details about the nature of the error as the message, which usually should be shown to the user.

8.7 Chunk and Annotations

Annotations (explained in the Glossary, chapter I) were initially created for Named Entity Searcher, which marks Named Entities such as Person, Place, Organization (described in greater level in chapter 5.3.5).

Later, we found out that the same classes can be elegantly used for other features of chunk, like dashes or newlines – because they are irrelevant for translation (core disregards annotations), but needed for rendering.

Another possible annotation usage for the future is subtitle formatting (bold, italic and underlined text). However, it seemed to be too complex, especially in the GUI, so we decided to not implement it in the current version of the application, because it is used very little and has no formal specification.

A Chunk has a set of methods to retrieve and set the annotations correctly, using the notion of forms. The forms used are:

- *database form*: the text as stored in the subtitle file, only cleaned – formatting stripped off (bold, italic etc.), non-printable characters turned into spaces, multiple spaces replaced by one space only and newlines are stored as PIPE characters.

- *surface form and annotations*: the surface form is the text to be used to generate the translation suggestions, without dashes and with newline turned into spaces
- *GUI form*: the HTML form to be displayed in the GUI. Newlines are turned into `
` tags.
- *text form*: the form to be used in HTML textareas – with dashes, newlines are generated as `\n` characters.

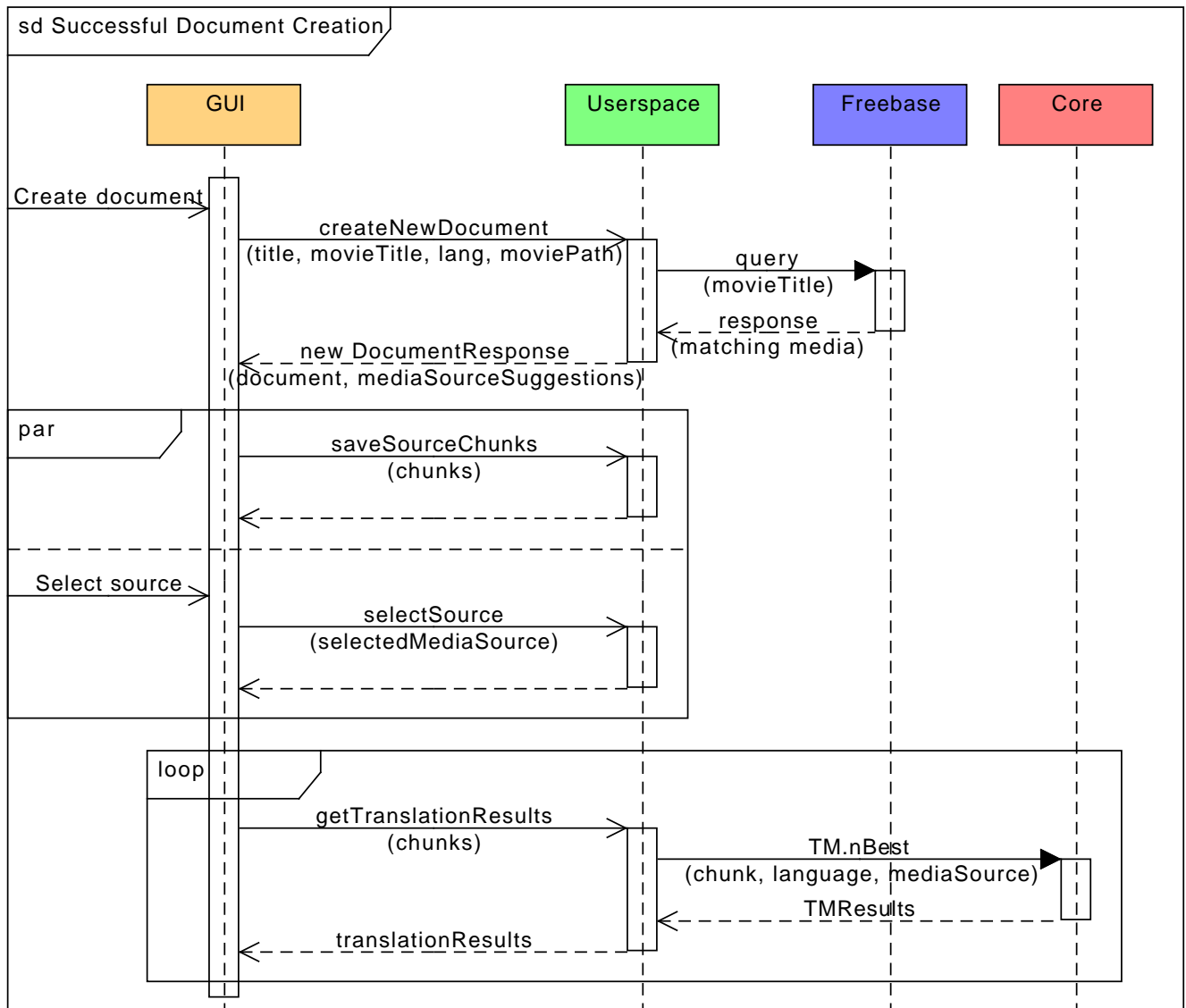


Figure 8.1: Sequence diagram of document creation, including translation suggestions loading.

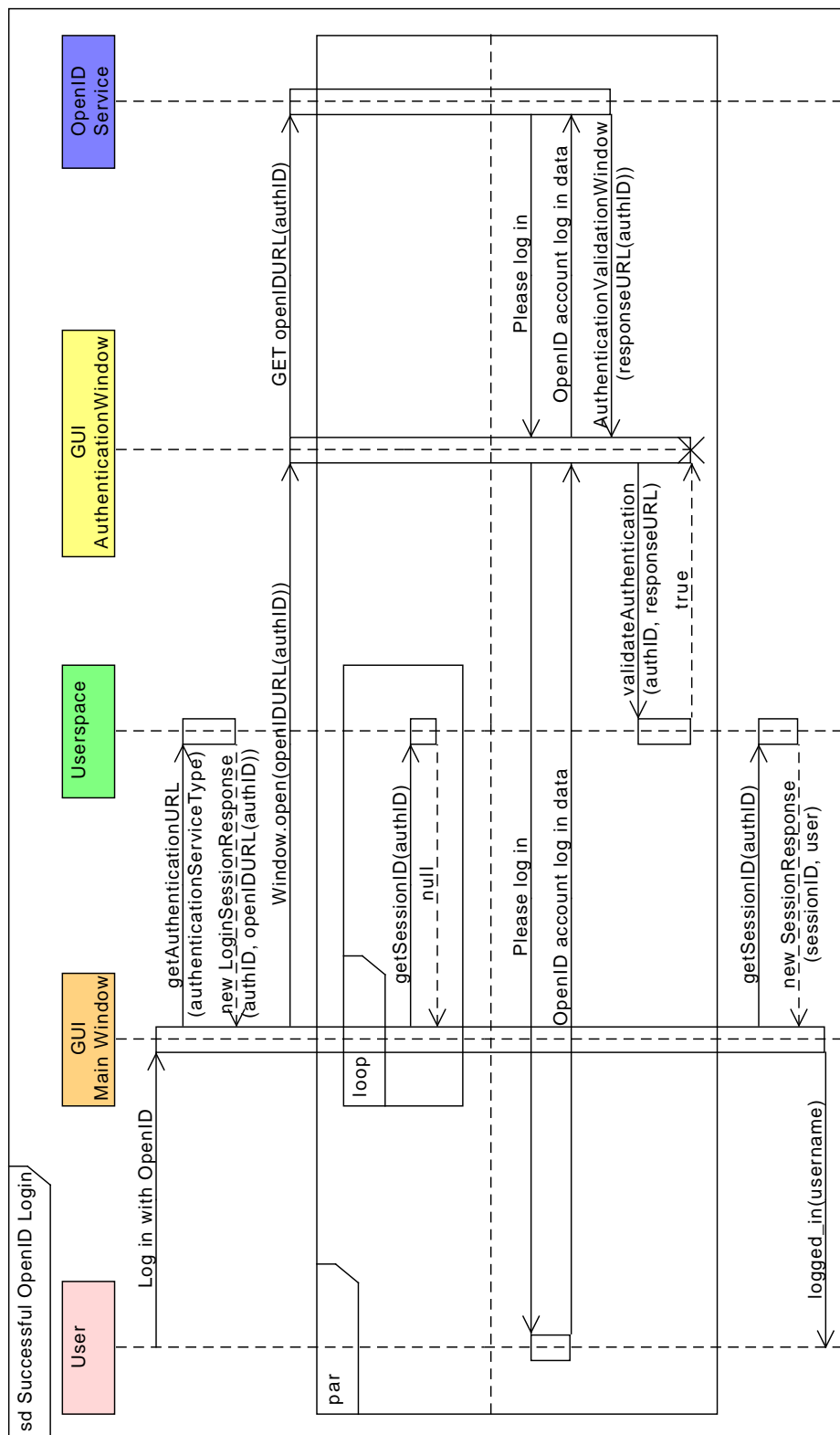


Figure 8.2: Sequence diagram of OpenID login.

Chapter 9

Graphical User Interface

The Graphical User Interface (GUI) is the part of the application that is loaded by the user in his web browser. It is written mostly in Java but compiled by the Google Web Toolkit (GWT) into JavaScript. Its main tasks are

- visualisation of the application and communication with the user in a user-friendly way, especially providing an efficient translation workspace adapted for translation of movie subtitles
- communication with the User Space (see Chapter 7) through the Remote Procedure Calls (see Chapter 8), especially to provide the user with translation suggestions from the core translation memory (see Chapter 5) and for data persistence

9.1 Goals

Our main goal is to offer a tool for the translators of the movie subtitles as easy and effective to use as possible. With the capabilities of the current web browsers, we decided to make FilmTit a web-based application, therefore sparing the user the need to download and install anything.

9.2 Google Web Toolkit

The web-based user interface representing the client part of the FilmTit application is based on the Google Web Toolkit (GWT) framework. This technology is a development toolkit for building and optimizing browser-based applications and is based on the idea of compilation of Java source code into JavaScript. Therefore, it enables us to write in Java even on the client side, but preserving most of the advantages of web application accessibility for the user.

The official description of GWT is as follows:

Google Web Toolkit (GWT) is a development toolkit for building and optimizing complex browser-based applications. Its goal is to enable productive development of high-performance web applications without the developer having to be an expert in browser quirks, XMLHttpRequest, and JavaScript. GWT is used by many products at Google, including Google Wave and the

new version of AdWords. It's open source, completely free, and used by thousands of developers around the world.

9.2.1 Reasons for Using GWT and Discussion

We decided to use GWT for several reasons. One reason is that it integrates very well with the rest of the application, which is written in Java. A prominent example is the communication between GUI and User Space. Not only is the interface defined by one simple Java class (`FilmTitService`), which is then used both by the GUI and the User Space, but thanks to GWT we are even able to use exactly the same classes (placed in `cz.filmtit.share`) in GUI and in User Space and send them easily through the `FilmTitService` interface. Thus, we avoided the trouble of having to keep each of these classes in two versions, Java and JavaScript, together with mappings to and from the transport format. (In fact, all of these do exist in the application, but only in the compiled code, completely hidden from the developer.)

However, there is a downside, which could have been expected but is difficult to manage properly. It is obvious that GWT does not support everything that exists in Java – instead, it supports only a subset of Java, with many GWT-specific additions. This is perfectly reasonable, and, as the subset supported is large enough, should not cause any problems. The IDE does not provide much support in this area, so often the only thing that a developer can rely on are Internet forums. Luckily, GWT is used by many developers and most of the issues that we encountered had already been solved by others (although sometimes the only solution was not a nice or clean one, such as having to implement some methods directly in JavaScript). Still, we spent much debugging time on problems that were actually problems of GWT that had to be worked around, instead of being problems of our code.

A requirement that we had for the web technology was to offer an IDE (in GWT realized by a plugin for Eclipse IDE, similarly applicable also for the IntelliJ IDE), possibly with debugging support. In GWT, there is a dedicated Development Mode for these debugging purposes, sparing the developer the need of recompiling all the GWT code for testing after each change. This proved to be very efficient in the first phases of the development when the GUI module was developed independently and, although it turned out to be quite difficult to set up the Development Mode once the GUI module was integrated with the rest of the project, it remained one of the most useful features of GWT.

Another option that we found useful is the possibility to directly insert real Javascript code into GWT methods. Although it is not very clean and cannot benefit from some of the features of GWT, it is sometimes the only way to do something that is supported in Javascript but unsupported in GWT.

An important point also was that GWT is freeware, or more precisely, is licensed under the Apache License, v. 2.0, with several third-party software libraries included in its distribution under other freeware licenses. It is even open-source, from which we benefited occasionally when examining the source code to understand some of its unexpected behavior, or even to extend some of its classes, overriding their behavior to suit our needs.

Although GWT can be integrated into a Maven project, it was far from flawless and a significant amount of time had to be spent on making everything work together, including maintaining

the IDE support. We managed to do that in the end, but we had expected this to be much easier. The main reason (but not the only one) is the specific directory structure, different for the Maven project and for the GWT project, which has to be adjusted in the configuration of both the Maven plugins and the IDE. And still after setting everything correctly, some features remained unavailable to some of us, e.g. running GUI in Development Mode using Eclipse IDE.

To conclude, we are happy that we decided to use GWT because of the benefits it brought to us, but we were disappointed by the number of complications that it brought as well. Were we to decide on a web framework to use for a similar project, we would still have to consider our decision carefully.

9.2.2 Browser Support and Optimization

GWT also provides a feature which makes the final web application similar in different major browsers and optimized to a certain degree for each of them. This feature is called *deferred binding* and it is based on generating different versions of the JavaScript code during compile time, only one of which needs to be loaded by a particular client browser at runtime. This process is by default maintained by the GWT compiler itself, so that the developer does not have to worry about it.

However, this “unification” is not complete, nor it is intended to be. It covers only the bigger and more crucial differences of the code behavior among various browsers, but does not address some slight variances e.g. in design of the basic elements (and their most common behavior). This seems reasonable, because forcing each browser to interpret the code in the exact same way would mean hardcoding almost everything from scratch and not using many of their provided features, which would be probably impossible anyway. Nevertheless, some of these “slight differences” which we have encountered proved to be quite crucial for our intended design (especially in the event-handling domain), so lots of work had to be done to reasonably unify the application’s behavior even among the major browsers.

Still, our hope is that GWT development will continue and will keep up with the development of web browsers, allowing us to provide browser-up-to-date versions of FilmTit simply by acquiring a new version of GWT and recompiling the project. Currently, the application works well with Opera v. 12 Firefox v. 14, Chrome v. 21 and Safari v. 5.1.5 or higher.

9.2.3 Designing by UiBinder

Another very useful and comfortable feature of the GWT for designing the user interface is the UiBinder. The idea behind this approach is to design the visual structure of the page (or its part) in an HTML-like way (this is called the *UiBinder template*) and its behavior and functionality in a Java class (called the *owner class* for the template). The UiBinder itself is then an object binding these two approaches together. This style of creating the web application supports the respected best practice to divide the visual design and the functionality. It also allows to create the web page’s appearance in a way which is more natural to web designers (e.g. HTML-like), as well as more readable and modifiable. Other advantages include supposedly better performance (as the browsers can better optimize the rendering from the UiBinder templates than from the heavier-

weight Java-based widgets and panels) and support for internationalization (not used by FilmTit at the moment).

The actual source code of a page designed this way (or of its segment) then consists of a file with the *.ui.xml Ui-Binder template (where also the style definition can be included directly) and a corresponding *.java owner class, where the elements of the template can be accessed as widgets (as well as made accessible to other classes). The owner class then can be simply instantiated and plugged into an existing design just like any other widget.

9.2.4 Twitter Bootstrap Library

For enhancing the visual appearance of our application to a modern look without a professional graphic designer, we have decided to use an existing open-source library for GWT which displays the page elements (and their groups) in the style of the Twitter pages. This library is called GWT-Bootstrap and is easily applicable with the UiBinder-style designing. It is also still a live project, so there is a hope that more features would be available in the possible future development.

Similarly to other third-party libraries used by FilmTit, the GWT-Bootstrap library is attached as a Maven dependency and downloaded automatically from its own Maven repository.

9.3 GUI Structure

The GUI is contained in the package `cz.filmtit.client` (not `cz.filmtit.gui` for “historical” reasons – “client” is the default package name in GWT for the client side of a web application). All of it is written in Java (in its subset that is GWT-compilable to JavaScript), with occasional methods with bodies written directly in JavaScript (for reasons clarified in 10.1.1). Thus, the GUI can actually only be compiled to Javascript.

The GUI also makes use of the shared classes (package `cz.filmtit.share`, see 3.2), which are written using only the intersection between standard Java and GWT-compilable Java and are therefore fully compilable both to Javascript and to Java bytecode.

Some settings have to be done via several resource files in the webapp directory. (This is required by the Java Servlets technology used to deploy the application.)

9.3.1 Gui.java

The main class is `Gui.java`. It defines the web application itself, providing an entry point, initialization methods, logging methods and access to page switching. It also contains fields that are considered global, such as the `sessionId` of the currently logged in user. It behaves as a static class, except for the `onModuleLoad()` method required by GWT, which corresponds to the `main()` function.

9.3.2 Pages

GWT – discussion

Very soon, we found out a weak point of GWT: it has nearly no native support for multiple pages. This was quite a surprising issue to us, since in any other web development technology or framework that we know, creating a new page and switching pages are really simple tasks. GWT, however, uses the idea of having one HTML file and one JavaScript file only that handle everything. (It is technically possible to create multiple pages by creating multiple GWT projects, each corresponding to one page, but this is not recommended for many good reasons.)

The GWT way of switching pages is by replacing the contents of the Root Panel (which represents the visible part of a page), or any other Panel placed within the Root Panel (which we decided to use so that some elements, such as the menu, are present on all pages). The switching is done without reloading, simply by creating an instance of the new page and directly setting it as the contents of the panel (without any convenience functions for that to our knowledge).

All the pages have the same URL by default, but GWT offers the History class for changing the URL and reacting to changes to URL (however, it is not linked to page switching by design – this remains for each programmer to do by himself). There are also Hyperlink widgets, which represent a link to a different page, with the default action being changing the URL and invoking the appropriate handler (a `ValueChangeHandler`). The URL contains an anchor with the name of the page (e.g. `#DocumentCreator` or `#WelcomeScreen`), and the History class is able to provide the name of the anchor to the user and to invoke the `ValueChangeHandler` when the URL changes.

We decided to use the History class and Hyperlinks to implement the menu and page switching. We create a `PageHandler` class which implements the `ValueChangeHandler` – it turns the anchor string into a value of `Page` enum, checks whether the requested page can be loaded (especially based on the user log in / log out state) and loads the appropriate page. It also provides public methods to other parts of GUI so that they can switch pages, modifying the URL at the same time (so that the browser refresh works as expected).

However, we encountered one issue with the Hyperlinks: the `ValueChangeHandler` is invoked only if the value of URL changes, which means that if the user clicks a menu link which points to the page loaded (e.g. while creating a document, the user decides to restart and clicks the document creator link), the handler is not invoked. As this behavior is hard-coded into the `Hyperlink` class, we had to abandon the Hyperlinks and used `NavLinks` instead (a simple link with no default interaction with the History), with click handlers invoking the `ValueChangeHandler`.

We also had a discussion about the expected page to be loaded in various situations. E.g. if the user is in Translation Workspace and logs out, the URL page parameter stayed on Translation Workspace by default, although Welcome Screen is displayed instead because a logged out user does not have access to Translation Workspace. If he then logged back in without switching the page, the document he had opened before logging out was reopened again.

Eventually, the behavior was modified so that the URL is always changed to Welcome Page when the users explicitly log out, but the URL is not modified if the users are logged out automatically, typically because their Session ID expire. Reasoning: if they were logged out implicitly, it is probable that they did not want that and would like to continue with their work as soon as possible. However, if they logged out explicitly, they probably finished their work and expects

the application to be “closed”, i.e. not remembering the last state of their work.

Structure

The subpackage `cz.filmtit.client.pages` defines the pages of the application. Please refer to the sitemap figure 9.1 for an overview of the pages, which are:

- `DocumentCreator`, used to create a new document
- `TranslationWorkspace`, the main page of the application where the actual translations take place
- `UserPage`, providing listing of user documents and the possibility to edit them
- `Settings`, enabling the user to change several settings, such as his password or the maximum number of translation suggestions to show
- `WelcomeScreen`, displayed as the first page of the application
- `Help`, showing the user manual of the application
- `PlayerInfo`, showing guidelines on how to run the media player
- `PlayerDemo`, showing a sample video for the user to test the media player
- `ChangePassword`, used as the target of the password change link sent by e-mail to users who forget their password, enabling them to set a new password
- `AuthenticationValidationWindow`, a special page that receives OpenID authentication data from an OpenID provider and passes them to User Space for validation
- `Blank`, a special page with no contents and is used to temporarily hide the contents of another page

As was already mentioned, each page has its layout and function separated as much as possible, with the function being defined in a `<page>.java` file and the layout in a `UiBinder <page>.ui.xml` file.

The pages package also contains the `GuiStructure`, which defines the content panel, into which the individual pages are loaded, and the top menu, which enables the users to log in and out and to switch pages.

A page is created and loaded by calling its constructor, which may require some parameters (such as `documentID` for `TranslationWorkspace`). Page loading and switching is handled by the `PageHandler`, which also provides support for URLs: e.g. the About page can be accessed by a URL `http://server/#About` or `http://server/?page=About` (the first one being the default), with the help of the GWT History class.

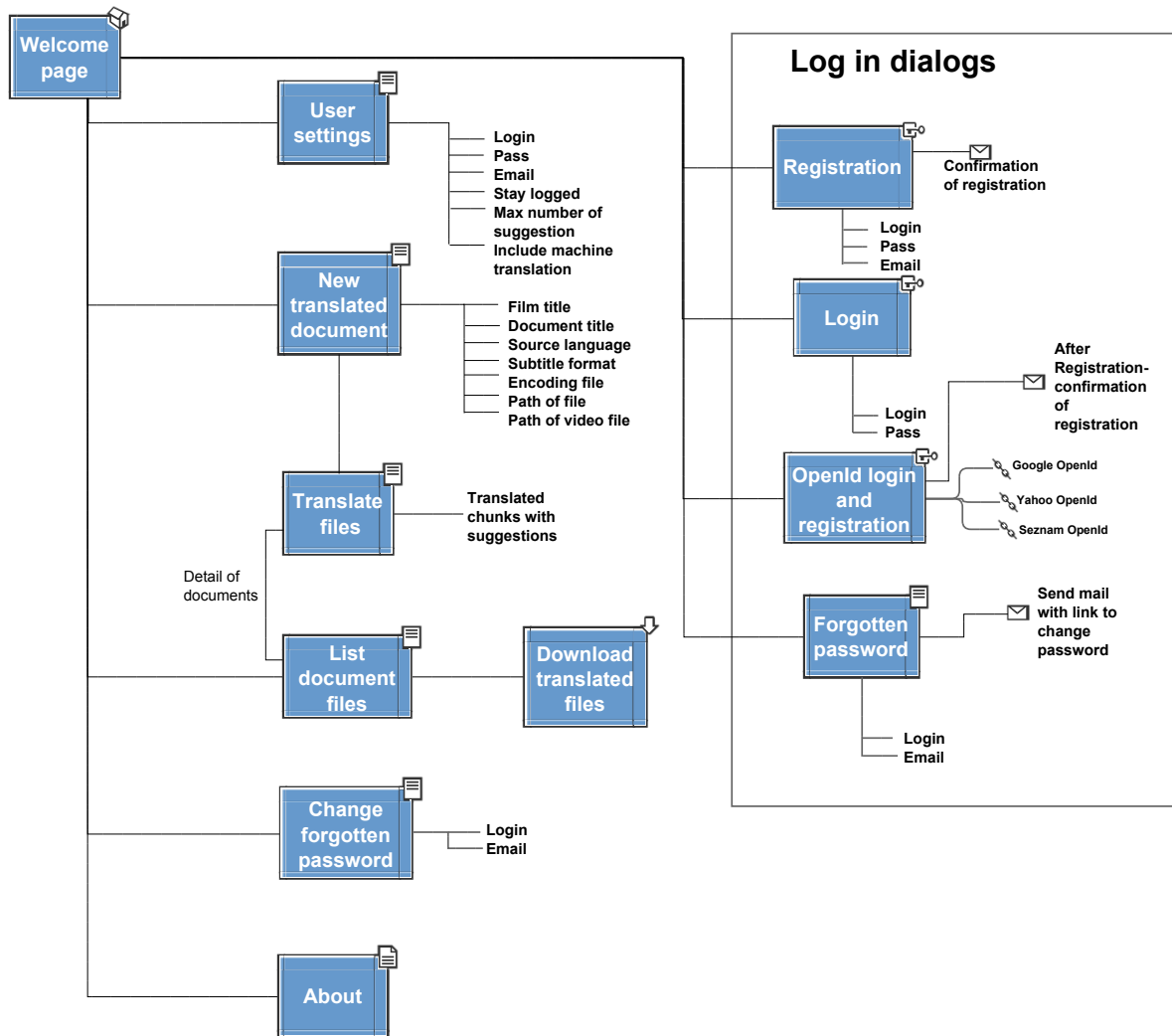


Figure 9.1: Site map of the application

9.3.3 Dialogs

Apart from pages, the GUI also contains several dialogs that can be displayed on top of a page. The dialogs are defined in the `cz.filmtit.client.dialogs` subpackage and are typically extended from the abstract `Dialog` class in the same package.

The `Dialog` class defines a modal dialog container and an “interface” to be used to handle the dialogs – all dialogs should only be accessed via this interface (except for their creation, which is invoked by their constructor). The interface methods are especially `deactivate()`, `reactivate()`, `showInfoMessage()`, `showErrorMessage()` and `close()`. To deactivate a dialog means to disable all of its active elements so that the user cannot interact with it and must wait for the dialog being reactivated or closed.

The Dialogs defined by the application are:

- `LoginDialog`, used for logging in and related functions,
- `SessionIDPollingDialog`, used when waiting for OpenID login process to complete,
- `MediaSelector`, used to select the correct movie or series of a newly created document or when changing the document’s movie title,
- `TimeEditDialog`, used to edit timing of subtitle items,
- `DownloadDialog`, which enables the user to download the translated subtitle file,
- `GoingOfflineDialog`, offering the user to turn on the Offline Mode, and
- `GoingOnlineDialog`, offering the user to upload data from Offline Mode when the user goes back online

9.3.4 Remote Procedure Calls Implementation

Each RPC is in GUI represented by a class from the `cz.filmtit.client.callables` subpackage. The name of the class is typically similar to the RPC name – e.g. for `deleteDocument` RPC it is the `DeleteDocument` class. Each of these classes is extended from the `Callable` superclass, which will be described in the next section.

The RPC is invoked by creating a new instance of the callable class, providing the necessary parameters in the constructor – e.g. to invoke the `simpleLogin()` RPC with the username and password parameters, one simply calls `new SimpleLogin(username, password)`. A reference to the instance usually does not have to be kept, since the actions to take on success or failure of the RPC are already hard-coded into the `onSuccessAfterLog` and `onFailureAfterLog` methods of the class.

Callable

The institute of the `Callable` superclass of all RPC classes has several purposes:

- to alleviate the burden of boiler-plate RPC invocation code by providing a wrapper for the “raw” RPC call

- to provide utility methods and default actions, such as error handling
- to provide actions to be always taken, such as logging of the calls with their parameters, and their results
- to provide a common structure for all RPCs

The Callable class itself implements the AsyncCallback interface, implementing the required `onSuccess` and `onFailure` methods; therefore, an instance of its subclass can be directly passed to the asynchronous call as the callback.

The subclasses representing the individual calls must only override the `call()` method, which specifies the actual RPC to be invoked. They can also modify the default behavior defined by the superclass by overriding several other methods, such as `onSuccessAfterLog`, `onEachReturn`, `onFailureAfterLog`, `onProbablyOffline` etc. This way, the behavior needed can be always achieved, but the default implementation is often sufficient, so the subclasses often override only one or two methods, keeping their code as simple as possible. (The method most often overridden is the `onSuccessAfterLog` method; its default implementation is to do nothing, which is only good for RPCs that do not require any reaction of GUI on their successful return, such as `changeDocumentTitle` or `stopTranslationResults`.)

Error handling

If the request fails for any reason, it is retried by default; four attempts are made for each request, always retrying after a short time interval. Resending the request is the default behavior which the subclasses override in cases where it is obvious that resending will not help (e.g. incorrect e-mail address format or an already existing username on registration).

In case of network problems, both temporary and permanent (this cannot be easily distinguished), the GUI usually receives a `StatusCodeException` with the status code 0. In such case the request is always resent three times before passing control to the `onProbablyOffline` method (this behavior cannot be overridden).

There is also a timeout for each request after which the request is regarded as lost and is retried.

The default action in case of an error is to show the error message in a JavaScript alert window, except for the `InvalidSessionIdException` where the default action is to ask the user to log in again. The subclasses often override this by showing the message in a Dialog (invoking the `reactivateWithError` method of the Dialog class), or by ignoring the error completely (e.g. when deleting a document, a failure with an `InvalidDocumentIdException`, meaning that the document does not exist, is an error, but there is no need to inform the user because the result is the same as if the RPC succeeded: the document does not exist now.)

Most of the RPCs contain the `sessionId` as one of their parameters to authenticate the user. All of such RPCs can thus throw an `InvalidSessionIdException`, to which the default action is to show the Login Dialog to the user.

9.3.5 Offline Mode

The Offline Mode offers the users the possibility to continue translating a document even without a connection to the server. Their translations are stored locally in his browser and sent to User Space once they go back online. Support for storing the data is ensured by the HTML5 Local Storage feature, which provides an in-browser data storage, accessible from JavaScript. Each object is stored as a pair of a unique string key and a string value (therefore, serialization to string is necessary to store complex objects).

Motivation

We decided that the only part that has to be able to work offline is the Translation Workspace, as doing the translation itself is the most time-consuming part of the whole process for a user. We believe that the user can prepare everything while online, wait until the translation suggestions are loaded, and then spend some time offline translating the document.

We also decided to require for the application to stay open while the user goes offline, as it would be difficult to reload the application offline – e.g. the whole application would have to be stored offline, which we wanted to avoid in the first place by designing FilmTit as a web application. However, it is obvious that the data created by the user in Offline Mode must be stored locally even if the application is closed. Thus, only storing `setUserTranslation` calls and invoking them once the user goes back online turned out to be sufficient.

We decided to use HTML5 Local Storage, which is similar to the well-known cookies mechanism from which it evolved. Like Cookies, the stored object has two “useful” string fields, the name (or “key”) and the contents (or “value”). However, the Local Storage has several advantages over cookies:

- The objects are not sent to the browser on loading the page but are stored and retrieved on request. This would actually be a disadvantage if we wanted to interact with the storage from the server side; however, in our case, being only able to handle the objects from within JavaScript is an advantage.
- The size limit both for individual objects and for the total amount of data stored is larger.
- Instead of setting expiry time, the programmer simply decides whether the objects should be stored temporarily (for a browser session) or permanently (which is our case).
- The API is cleaner.

Still, thanks to the similarity of Local Storage to cookies, it would be quite easy to implement a fallback for browsers without Local Storage support using cookies. We decided not to do that because, as far as we know, most browsers installed on users’ computers nowadays should support Local Storage. However, we might add this feature in future if there is a need for it.

Implementation

The Offline Mode support is separated into three parts: the `LocalStorageHandler`, the `Storable` interface, and classes implementing the interface (i.e. the `SetUserTranslation` class).

Storable interface

This interface defines methods that need to be implemented by a class to be storable in the Local Storage, especially:

- `toKeyValuePair()` – to serialize the object into a pair of a key and a value
- `static fromKeyValuePair(KeyValuePair)` – a factory method that deserializes the object from a pair of a key and a value (not actually defined by the interface; for a discussion on that matter see Section 10.1.4)
- `onLoadFromLocalStorage()` – invoked by the `LocalStorageHandler` when the user decides to upload the object

Each implementing class defines its serialization such that its key uniquely identifies the object and the value contains all data (except for those already contained in the key) needed to reconstruct the object. (The key and the value are expected to consist of semicolon separated fields by default, but each class can define its own serialization, there are no restrictions other than defined by the `String` class.)

To store in the Local Storage, the key is extended to a “full key” by adding the `classID` (e.g. “`SetUserTranslation`”) and the `userID` (a long), so the resulting format of key is:

`full key = userID@classID:key`

When loading the object from the Local Storage, these added fields are used to identify the owner of the object and the class to deserialize the object into, and are stripped before passing the key to the `fromKeyValuePair()` method; thus, the class gets the key for deserialization in exactly the same format as it was produced by the class on serialization.

Unlike the key, the value is defined solely by the implementing class and is stored and loaded “as is”.

The `Storable` interface is currently only implemented by the `SetUserTranslation` class.

SetUserTranslation

The `SetUserTranslation` class defines its serialization key and value as follows:

- `key = documentId;chunkId;partNumber`
- `value = chosenTranslationPair;userTranslation`

The `onLoadFromLocalStorage()` method implementation invokes the `setUserTranslation` RPC.

LocalStorageHandler

The static `LocalStorageHandler` handles storing and loading of `Storable` objects to and from the Local Storage.

An important field is a boolean “online”, which determines whether user is in Online Mode or Offline Mode. Setting this value is equivalent to switching the Offline Mode on or off.

The `LocalStorageHandler` implements especially the following (static) methods:

- `storeInLocalStorage(Storable)` – takes a `Storable` object, serializes it and stores it into the Local Storage
- `loadUserObjectsFromLocalStorage()` – examines the Local Storage and returns a list of objects that belong to the current user
- `uploadUserObjects()` – deserializes the loaded objects and invokes their `onLoadFromLocalStorage()` methods

The Offline Mode operation

Please see the sequence diagram 9.2 for the first phase, where the user goes offline and continues working on the translation, and the sequence diagram 9.3 for the second phase, where the user goes online again and the locally stored data are uploaded to the server. (For simplicity the diagrams do not show some implementation details; also, parameters are listed only if they are necessary for understanding the process.)

Before using the Local Storage, it is checked whether it is supported – if it is not, Offline Mode is not offered to the user and a relevant message is displayed.

When a `setUserTranslation` call fails with the “probably offline” error, it is saved into a queue of failed calls that are to be stored offline if the user agrees (other `setUserTranslation` calls could already have been invoked and will also fail and get enqueued). If the user decides to turn on the Offline Mode, all calls from the queue are stored into Local Storage.

In Offline Mode, the style of the menu is changed and all active components are hidden, because the user should not leave the Translation Workspace while in Offline Mode. Also, going back or forward using the browser controls is disabled: each upcoming page switch is cancelled.

The `SetUserTranslation` calls invoked while in Offline Mode are not sent to server, but they are serialized into a key-value pair and stored in the Local Storage.

Once the user is back online and logs in again, the `LocalStorageHandler` examines the Local Storage. It strips the `userID` part of the key of each object found and compares it to the currently logged in user. If some data belonging to the users are found, they are informed about their count and are suggested to upload the data. If they agree, `LocalStorageHandler` goes through the data, stripping off the `ClassID` and invoking the corresponding “`ClassID.fromKeyValuePair()`” deserialization. Subsequently, the `onLoadFromLocalStorage()` method is invoked on each of the resulting objects – the `SetUserTranslation` implementation of this method is to invoke the `setUserTranslation` call, i.e. to store the translation on the server. When the call returns, `LocalStorageHandler.SuccessOnLoadFromLocalStorage()` or `LocalStorageHandler.FailureOnLoadFromLocalStorage()` is called, as required.

When each of the loaded objects has either succeeded or failed, the user is informed about the result. In case of errors, he can decide to retry loading the failed data or to delete them.

9.3.6 SubgestBox

An important class is the `SubgestBox`, or “SUBtitle sugGESTion BOX”, which provides a textbox-like interface and visualizes the TM results, offering a variety of means of navigation through

them. It is based on the IFrame HTML element to support also multi-line and formatted inputs. The TM results are shown as a pop-up suggestion list when the textbox is focused. Another features like auto-scrolling to a certain place of the screen and height auto-adjustment for multi-line inputs were added to improve the user experience. The behaviour and features mentioned requires a custom event handling, this is provided by the SubgestHandler class (common to all the SubgestBoxes within a Translation Workspace).

Because of performance reasons, the SubgestBoxes in the Translation Workspace are not initially all loaded as IFrames (since it would cause a heavy load for the browser to handle), but as simple text-areas instead. The conversion to the “real” SubgestBoxes only takes place at the moment of their first focusing. The preliminary text-areas are represented as objects of the Fake-SubgestBox class, inner class of the SubgestBox.

9.3.7 Playing the Movie Files

From the beginning, we wanted to be able to show the user the movie file along with translations. However, legal issues, bandwidth issues and disk issues bar us from actually uploading the files to our servers – so, we tried to implement a browser-based player for the users’ movie files. Users should be able to play movies from their disks in the browser, and the translation workspace should be able to communicate with this player via JavaScript.

For this, we have to deal with two problems.

Video playback (VLCWidget)

Playing the movie locally, from disk, is actually the lesser problem. We use the VLC browser plugin for that.

VLC¹ is a popular open source media player. One of its advantages is that it can handle almost any video format and any popular codec available.

VLC has also a feature called “browser plug-in”². The browser plug-in is not a part of the standard VLC installation, but can be installed easily.

This browser plug-in can be used for playing local files in the browser, given that the file address is known. It is possible to rewind forward/backward/stop, all through JavaScript.

There are some slight difficulties in the plug-in, though. One of them is, that it is not possible to jump on an exact time location in the video file. Most video files have so-called seek frames, which can appear more or less often (depending on the setting of the encoder), and the VLC plugin always moves to the closest preceding seek frame. The difference can be quite big (even up to 20 seconds). This bars us from playing exactly the time slot of a given subtitle.

The other difficulty is that sometimes, the plugin just freezes without any warning after a move to a position. We have found out that it is somehow connected to the files played; with the exact same movie files, the exact same positions cause the player to crash, but slightly moved positions are not problematic.

¹<http://www.videolan.org/vlc/>, short for VideoLan Client

²For some reason, its name is different across the VLC documentation. Sometimes, the name “Mozilla plug-in” is used, sometimes “Web plug-in”, sometimes “browser plug-in”.

The first issue is solved by not playing just the part with one subtitle, but bigger, 30 seconds “windows”. The entire movie is split into these 30 second windows, and once the user switches from the subtitle in one window to a subtitle in another window in the TranslationWorkspace, the whole new window is played. It is actually better than playing just the time of the subtitle, since the user has more context to translate the movie’s sentences.

The second issue is resolved by detecting the crashes and restarting the whole plugin in the case of a crash with slightly moved windows.

The entire player is implemented in `cz.filmtit.client.widgets.VLCWidget`. Apart from that, the `VLCWidget` displays the subtitles on the left and on the right of the VLC browser plugin (original subtitles on the left, translated on the right).

We intentionally did not implement very sophisticated controls for the player (since it would make the design too complicated). The functionality of the player is restricted to replaying the whole current window or pausing the playing.

Java applet for reading the filename

The bigger problem is actually *reading the name of the file* before giving it to VLC player plugin.

Modern browsers are generally built to give websites as little access to user’s system as possible (and understandably so!). These restrictions include the file paths of uploaded files. Even the recent HTML5 standards that do have access to the file itself cannot read the filename.

The only way we found that would be usable across web browsers and operating systems, was to use a Java applet. Java applets have access to the user’s disk and can read the filenames; they have to be signed by a certificate, but that certificate can be self-signed.

This is implemented in `cz.filmtit.client.widgets.FileLoadWidget`.

The problem with this approach becomes evident – we are now dependent on two external, non-standard, separate plug-ins for the task of playing video files – the VLC plugin and the Java plugin. Both have to be installed separately, both often cause issues in various browsers (not only in theory). On the other hand, we achieved the almost impossible task – playing movie files from user’s disk in their browsers. If we therefore warn users that this functionality is experimental and carefully explain the two plugins and their installation, we believe the current implementation is acceptable.

9.3.8 Parsing and Segmentation

We already introduced the parsing in Chapter 4.3. As we mentioned there, we are reusing the classes of the `dataimport` module (where we parsed and segmented thousands of files) in the GUI.

More information about the parsing can be found in Chapter 4.3.

9.3.9 Manipulating with the Documents and Subtitle Items

The user is provided with the options to change some of the properties of the given document or subtitle item. Within the document listing, they can change the title of a document or the corresponding movie without affecting its contents. Also, in the Translation Workspace, a subtitle

item can be edited in more ways than only inside the textboxes. The timing can be changed, which is a useful feature especially when used with the embedded player. The user is also allowed to change the source text, this is used typically when there is an error in the original subtitles, because that could deteriorate the quality of the suggestions for this item.

We do not offer explicit deleting of the chunks; however, the default behavior is that the untranslated chunks are not exported, so not translating a chunk is roughly equivalent to deleting it. We also do not allow to change the source subtitle file (this would actually mean deleting the whole old document and creating a new one, probably only keeping the title – so we leave this for the user to do himself), although smaller edits can be done directly in Translation Workspace (see above).

Most of the non-textbox editing are accessible via clicking or double-clicking on the corresponding text or number. When there was a doubt about the intuitiveness of such action, a tooltip is provided as a hint for the user.

9.4 Logging

In User Space and Core, we use *Log4J*, which implements the standard Java Logging interface for logging.

The situation with the GUI is not so easy. GWT provides some default logging tools which work in GWT development mode, but not in production. (As a temporary solution, we used a debugging console in GUI, but it is turned off in the final version.)

The final solution we wrote is what we call “remote logging”. Important messages are now sent to the server through an RPC, where they are both included into the server log and saved into the database for possible future use (together with the userID if possible).

The minimal level of a message to be sent to the server can be set (debug, info, warn, error). We log important GUI events and exceptions with their stack traces in order to be able to react on potential bug reports later.

There were some technical difficulties – mainly catching unexpected exceptions in GWT and umbrella exceptions. An umbrella exception is a GWT exception thrown when multiple exceptions have occurred at once, and it contains all of them. Its message and stack trace are not useful at all because they only cover the creation and throwing of the umbrella exception itself, but do not reveal anything about the inner exceptions.

9.5 Typical Usage

9.5.1 Document Creation

In the most typical case, users (already registered) will open the page with FilmTit and log in. Then, they will go to the DocumentCreator page to create a new document. A document corresponds to one movie, or more precisely, to one subtitle file being translated from one language to another. On this page, the users should specify the title of the movie (or series), the direction of the translation (e.g. from English to Czech) and provide the actual local file with the

source subtitles. They can also set a document title different from the movie title, and set a file path to the corresponding movie video file on his file system. These information are sent to the User Space and, after refining the movie specification through the `MediaSelector`, the new document is created and the page switches to `TranslationWorkspace`, filled with source subtitles (segmented to chunks) together with their timing on the left side and corresponding empty translation text-boxes on the right side.

See also the corresponding sequence diagram 9.4.

9.5.2 Document Translation

At the same time, the subtitle chunks are sent through the User Space to the Core, where the appropriate translation suggestions are generated for them and sent back. When received in the GUI, each translation result is set to the `SubgestBox` corresponding to its chunk, and can be displayed on activation of the translation text-box as a pop-up, with the suggestions sorted according to their estimated accuracy. The user can then choose any one of them (or none) and edit it (if necessary) to get the translation of the particular chunk. When the users leave the translation text-box, their translation is sent to the User Space through the `setUserTranslation` call to be saved as the user translation of the chunk.

See also the corresponding sequence diagram 9.5. (Some technical details are ommitted for simplicity.)

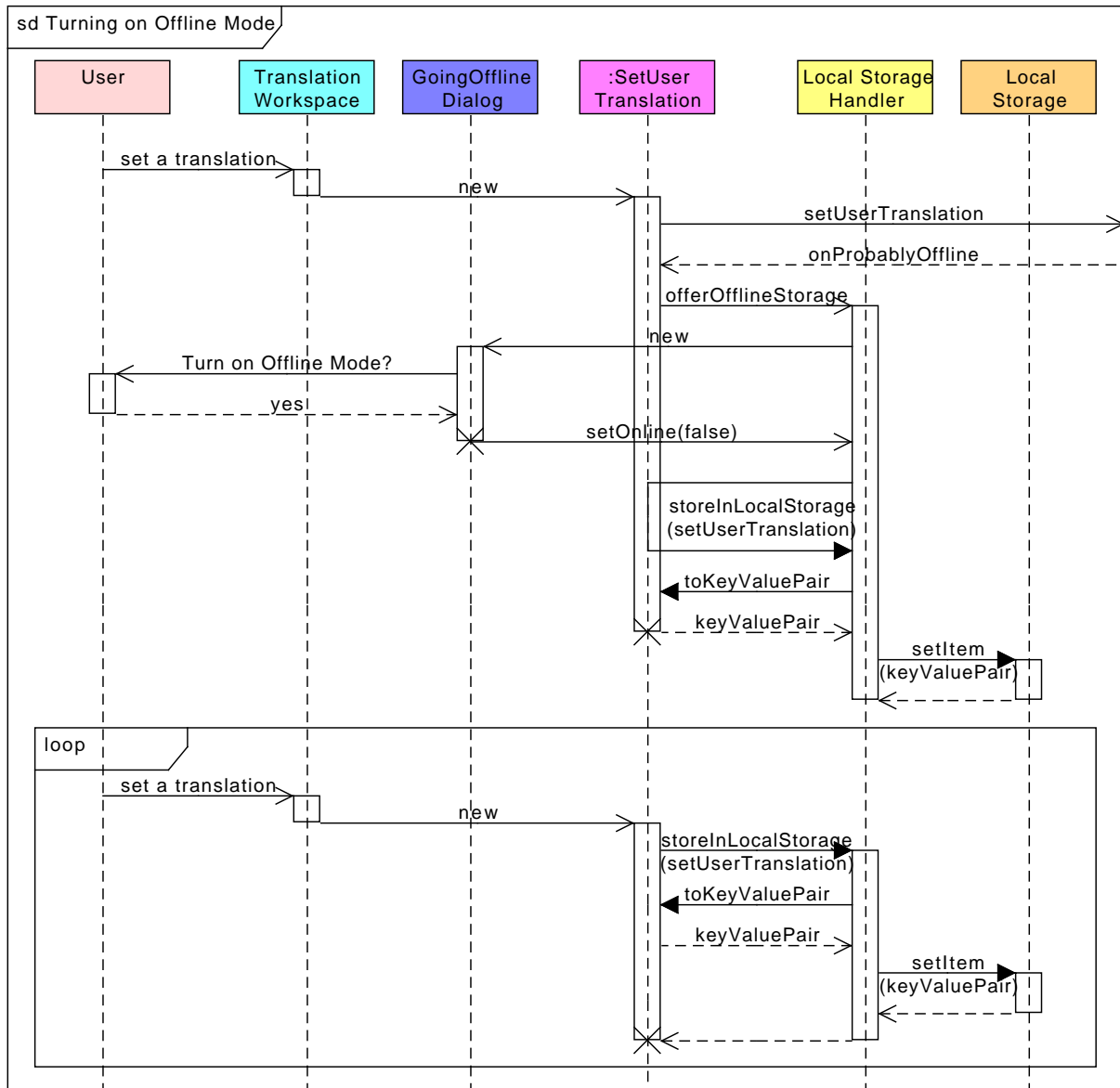


Figure 9.2: Sequence diagram of turning on the Offline Mode and translating the document in Offline Mode.

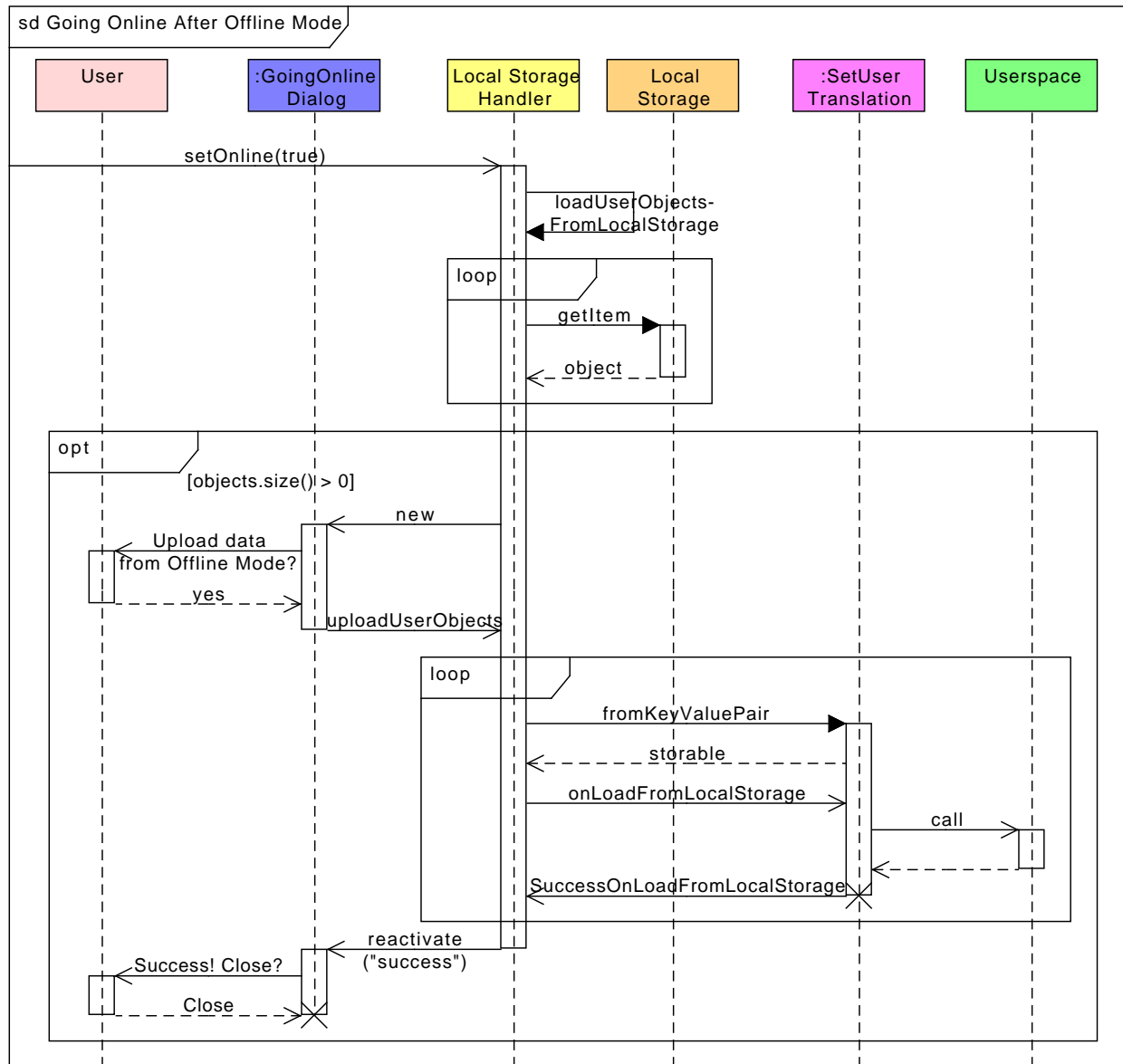


Figure 9.3: Sequence diagram of the user going online after using Offline Mode.

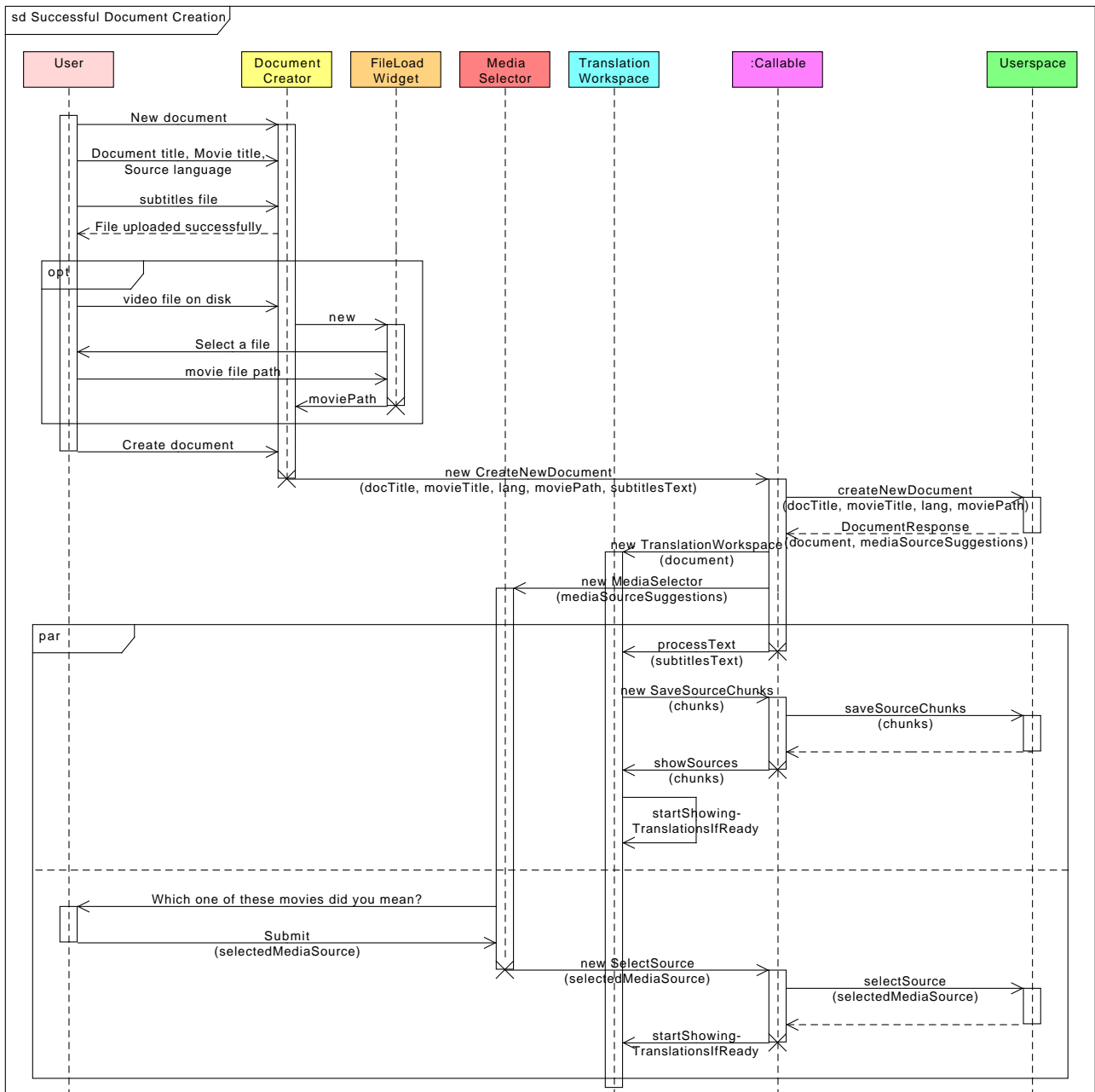


Figure 9.4: Sequence diagram of document creation.

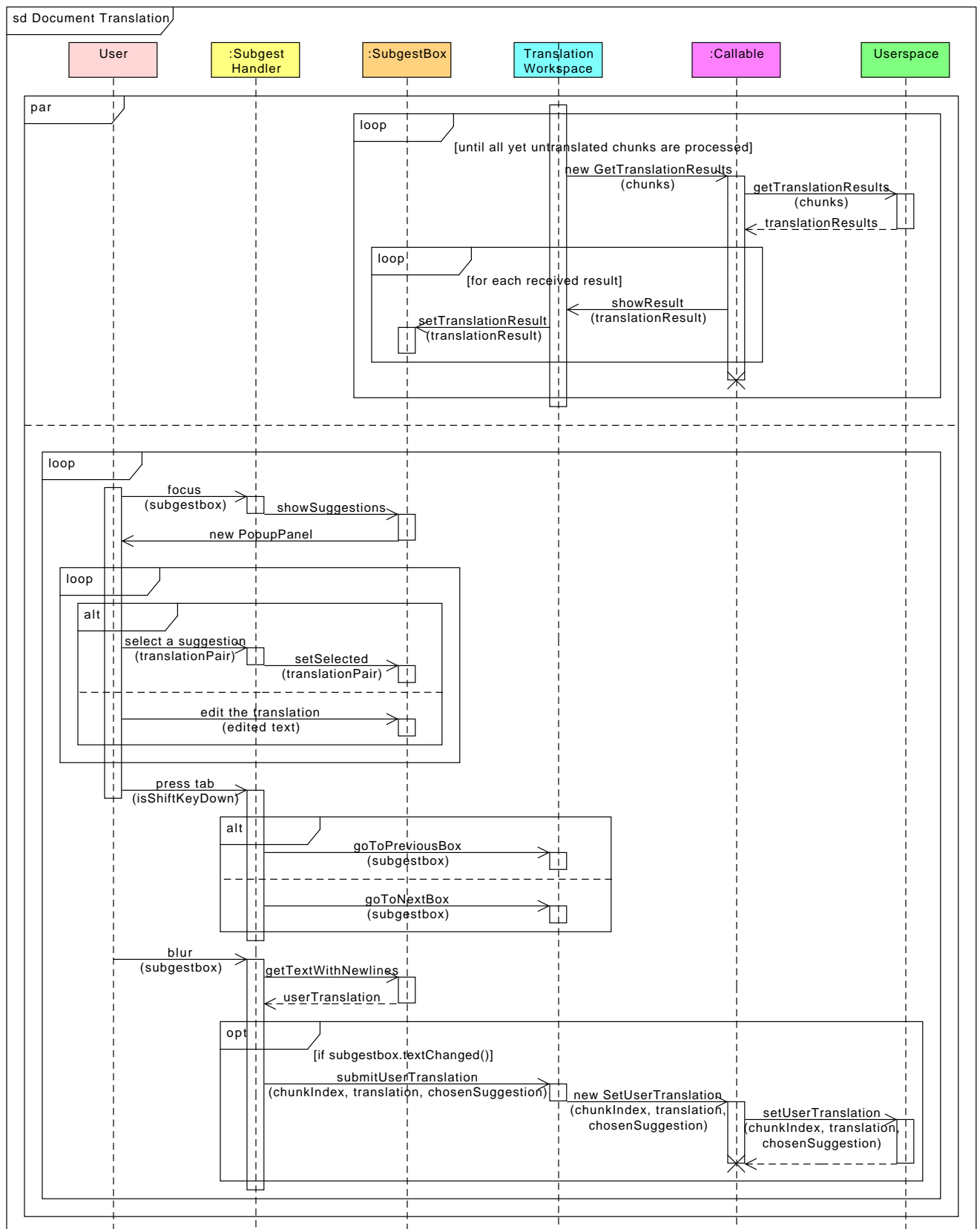


Figure 9.5: Sequence diagram of document translation.

Part II

Development Process Documentation

Chapter 10

Implementation and Development Process

10.1 Timeline

The following section describes the project development, progress and decisions we took through the time the project was developed. After a series of informal meetings, the project officially started in February 2012 and was submitted in September 2012.

10.1.1 Initial Project Meetings and Early Implementation Decisions

The regular project meetings where we discussed mostly the organizational aspects and the top-level design of the application started in December 2011. We easily agreed that we were aiming to create an application quite similar to Google Documents. We have decided very early to use a multi-tier architecture consisting of

- core translation memory,
- user space,
- graphical user interface,

which became very soon separate Maven modules, as we soon started using Maven for building the project.

After overcoming the problems connected with learning the new technology, we became quite satisfied with the decision to use Maven, because it enables to easily combine the parts of the project.

Project structure

Despite the fact that we added further modules later, we consistently kept the initial project separation into *Core*, *User Space* and *GUI*. At the beginning, we also assigned team members to different parts of project, which remained relatively stable as well.

Initial data

Before receiving the data from OpenSubtitles.org, we were thinking about the source of data to fill the translation memory for the first time. There were several options – either using the subtitle part of the Czech-English parallel corpus CzEng developed at ÚFAL¹, using sentences from a general purpose parallel corpus or getting the data from a subtitles server.

Programming languages

From the very beginning, we intended to base the project on Java, mainly because there are a number of web technology projects based on Java and every team member was familiar with the language. We also decided to combine the code in Java and Scala programming languages. At that time, there was only one team member who knew the Scala language. Although we repetitively expressed believes that everyone of us would learn it, at the end there was only one other person familiar with the Scala language.

The decision to use both Java and Scala appeared to bring a number of complications. On one hand, Scala allows to write concise and efficient code; on the other hand, most project members were not able to learn Scala sufficiently even to understand the Scala code properly, let alone actively producing Scala code, and only used Java. Although the interoperability between Scala and Java works well in most cases because both are based on the JVM, some problems remain. One of the problems of interoperability was, for example, that a *List* object created in Scala is not compatible with Google Web Toolkit, which expects a standard Java *List* implementation.

Technology for the Client: Google Web Toolkit

Much more complicated to agree on was the technology of the client. There were many different opinions, from writing the client in *PHP* with *Nette Framework*, which some of us knew well, to using the *JSP* to have all the code consistently in Java, even to quite an extreme idea to make the whole application a *Java Applet* (this idea had appeared because of the intention to integrate the video player in the application, since at that time we did not know any other way to do that except creating a Java applet). Finally, we decided to use *Google Web Toolkit*, which no team member had any prior knowledge of, but it promised making the communication between server and client and many other things very easy.

More information about the decision for using GWT and a discussion of its advantages and disadvantages can be found in Section 9.2.1.

10.1.2 Early Development Process

Luckily for us, we very soon received a database from *opensubtitles.org* containing all the Czech and English subtitle files that were at the server at that time. Soon after that we started working on an alignment algorithm to retrieve the parallel data from the subtitle files (the process is described in Section 4.5) and enable us to start experiments that helped us to decide which database system could be used.

¹<http://ufal.mff.cuni.cz/czeng/>

Database system

Choosing an appropriate database system was also an intensively discussed issue. The database underlying the Translation Memory plays a crucial role for the whole system performance. Also using built-in features could save us a lot of additional work. We evaluated different DBMS and decided to use Postgres (see Section 5.1.1 for details). To test its applicability for our project, we ran several small evaluations of Postgres features that might be useful to us, e.g. the full-text search. As soon as we made the final decision on which DBMS to use, we refactored the evaluation code into `TranslationPairSearcher` classes and started to set out the general architecture of the core. The `BackoffTranslationMemory` class was added and the candidate search was finished in its early stages, so a basic version of the system could be used even though there were no sophisticated rankers yet. For more details on the core architecture, please see Section 5.2.

Not having any experience with the object-relation mapping libraries in Java, we decided to use *Hibernate*, based on advice from both Internet forums and some of our colleagues.

Data alignment

Originally, all the algorithms processing the data we retrieved were implemented in Perl, and the code for importing the data was in the *Core* module, implemented in Scala. Later, to make the code more consistent, we decided to move the data preparation and data import to a separate *dataimport* module and to re-implement the Perl scripts in Scala.

Timing	Original chunk	Match	Suggested translation
00:00:54,542 - 00:00:56,965	This was the previous chunk.	Toto byl předchozí titulek.	
00:00:57,215 - 00:01:00,670	This is a subtitle chunk.	There is a chunk.	Tam je kousek.
			Teď tam byl.
		This is subtitle file.	Jsou to titulky.
			Titulky.
		This is punck !!!	Známka puncku !!!
00:01:02,847 - 00:01:05,836	This is the next chunk.		

Figure 10.1: Scheme of the originally intended structure of work with the translation memory. It reflects the original User Space structure and also schematically the original client design.

Video playback

Very soon, we started to try to solve the video playback in the browser, which we expected to be a really challenging issue. We did some research about *Adobe Flash* technology. We were also thinking about creating a hybrid solution – an application wrapper with a web browser inside, capable to ensure the video playback for the inner web application.

We very soon had toy implementations of:

- a player using VLC plugin, to be incorporated into the web application
- a player as a desktop application, showing the web application in a frame (see Figure 10.2)

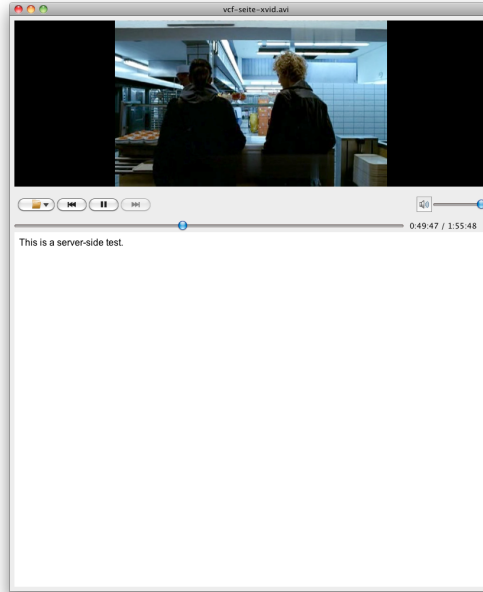


Figure 10.2: Screenshot of Qt-based desktop application.

We decided for the web application-only version, with no desktop variant, because, observing the development trends, it seems to us that the present and future of applications is in web applications. The benefits are e.g. that the learning curve is typically better for a web application, where the user is already familiar with most of the controls and work patterns, installation is not required, so anyone can start using the application immediately, it is easy to use OpenID registration.

Still, there are disadvantages, especially the cross browser support issues, limited power, and larger bandwidth consumption. An added advantage of GWT is that the bandwidth consumption is actually closer to that of a standalone desktop app, because the application itself is stored in one JavaScript file, which is downloaded at the beginning (similar to installing a desktop application, but it is done transparently), and the rest of the client-server interaction is done through RPCs (basically the same way as it would be done in a desktop application).

An issue we encountered in developing the player and were not able to combat completely is letting the user choose a media file and passing its full file system path to the player. We found out that although it is quite easy to load a whole file from disc into memory, only its filename is passed to JavaScript instead of the full path for security reasons. We did a thorough search on the Internet but found out that there is most probably no clean way around this restriction.

Ultimately we came up with 3 solutions, but none of them pleased us enough:

- the user inputs the full path as a string into an input box; works well but is not user-friendly (most users probably do not even know that such a thing as a file path exists)

Timing	Original chunk	Translation
00:00:54,542 - 00:00:56,965	This was the previous chunk.	Toto byl předchozí titulek.
00:00:57,215 - 00:01:00,670	This is a subtitle chunk.	
		Tam je kousek. <i>There is a chunk.</i>
		Jsou to titulky. <i>This is a subtitle file.</i>
		Titulky. <i>This is a subtitle file.</i>
		Známka puncku. <i>This is punck !!!</i>
		Toto je titulku článek. <i>Machine Translation</i>
00:01:02,847 - 00:01:05,836	This is the next chunk.	

Figure 10.3: Current scheme of the work with translation memory.

- the user chooses the file in the file browser and then copies the file path from the browser input box into a textbox; this is probably even less user-friendly
- a Java applet is loaded to choose the file as Java applets do not have such restrictions as JavaScript, and then passes the path to the application; it seems to us too heavy-weight to create and load a whole applet only to get the file path, but it works quite well – except for the users who do not have Java installed and working properly (we have also encountered an alternative using Flash for the same purpose, but we still prefer using Java to Flash)

Despite having its disadvantages – it is not particularly fast and from the users' view it could be considered not be really safe – we decided for the third option.

10.1.3 Introducing the Shared Classes

After having implemented a very basic version of all three parts of the project, we decided it was time to start to solve the interoperability of individual parts in order to run a first snapshot of the application. This was happening approximately in March 2012.

At this stage we found out that we are not fully taking advantage of using the Java technologies for all parts of project and decided to totally redesign the User Space and Client parts.

Originally, we wanted to keep the traditional translation memory structure where each sentence can have several matches and these matches can have several translations in the memory, as is depicted in Figure 10.1. However, this scheme did not reflect much the way we worked with the data in the Core, moreover there were not much cases where there would more translations for one match.

Also, the design of the GUI was a little bit confusing because the text of the matches was placed more prominently than the translation suggestions, despite the fact that the user is probably much more interested in the actual translation suggestions than in the matches. This lead to a modified scheme which is depicted in figure 10.3.

We agreed on the shared classes that all parts of the project should use. We more or less adopted the design of the classes from the core and started to use them in the whole project. This step required to re-implement some Scala classes in Java and to drop code that was already done in the User Space and the client. The design of the classes was almost the same as the final shared class design.

From a later point of view, it appears to be an important decision to agree on the shared classes, which made the cooperation between the modules easier and less verbose.

GUI layout

We started the work on the project with the new design soon, which lead to a period of struggles with technologies. It took us almost two months to have the first running version of the application.

The very first version of the application was a page where it was only possible to upload a subtitle file and to do the translation, without any possibility to load an already saved subtitle document or download the result of the translation, without any sessions or users; it was just a page where you edit the subtitles (which later became the Translation Workspace).

External APIs

At that time we used machine translation from the MyMemory service (which in fact wraps Google Translate), but there is a limited access per IP address and we soon began to reach the limit very frequently. It became obvious the we would have to change the source of machine translation. Based on that we decided to train our own Moses system. See Chapter 6 for details.

We also used an API providing IMDB.com information to receive information about movies but the movie meta data was not used in the evaluation the matches at that time. We had to switch to Freebase later because the IMDB service we used was discontinued.

10.1.4 The Main Development Phase

It is hard to define the main development phase which is covered in the following paragraphs. It corresponds to the period from the beginning of March when the important design changes were made (see previous Section 10.1.3) to approximately the middle of August when adding new features was stopped (see the next Section 10.1.5).

The section is divided to parts covering the issues we were dealing with and describes what we did in each area in those approximately five and half months.

OpenID

Although we wanted to implement OpenID support from the very beginning, we started with it approximately in May. We found the two most frequently used Java libraries for OpenID and tried them. Those were *JOpenID* and *openid4java*.

JOpenID seemed to be easier to use and also was much smaller as a dependency; on the other hand, when we ran into some problems with using it, we found out that *openid4java* is much

more frequently discussed on the *StackOverflow.com* forum and that there is a bigger chance that we would be able to find some advice for potential problems.

However, as discussed in 7.5.4, *openid4java* is easier to implement, so we used that, even for its biggest shortcoming – users are not able to set their own OpenID provider.

The support for OpenID was finished in July, the parser for the authentication data from Seznam was added at the beginning of August.

Chunk Splitting

An interesting issue, also from the linguistic view, we had to solve was to find the best way of splitting the subtitle items into chunks. As opposed to the classical way of simply splitting documents into sentences, we had several possible ways of doing the splitting because of the nature of subtitle files.

We talk about chunk splitting more in chapter 4.4.

We then realized that it is important that the splitting the same when importing subtitles into the database and when processing the user subtitles (as discussed in 4.3.2), so we made it a shared class.

Logging GUI

After a while we found out that logging GUI errors is as important as logging userspace and core errors. We describe the GUI logging in section 9.4; we first had a temporary debugging console, we switched it to remote logging mechanism at the beginning of August.

Implementing the RPCs

From the start, it was obvious to us that we want to use a well-established method to implement the Remote Procedure Calls. We noted in the specification that we would probably use JSON for the message serialization. Fortunately, GWT does provide a ready-to-use Remote Service implementation, which is described in bigger detail in chapter 8.1. GWT, as a matter of fact, does actually serialize the messages to JSON internally.

Sending Translation Results

What we have been almost continually changing was the way the results are requested and then sent back by User Space. The discussion is described in greater detail in Section 8.2.1.

User Management

At the very beginning the GUI was basically only the translation workspace. It was possible to translate a subtitle file in the application, the changes were reflected in the database, however it was not possible to get them in the application when the web application was reloaded or to export the subtitles.

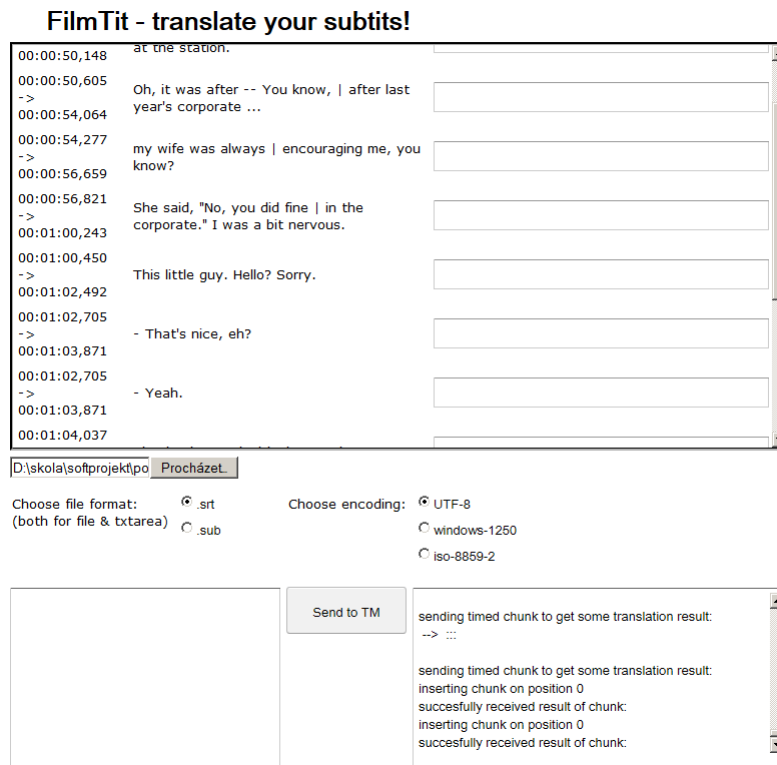


Figure 10.4: Design of the Translation Workspace before starting to use the Twitter Bootstrap

We considered the user management to be just a minor technical issue. However, to enable to reach the intended architecture of the User Space – mostly that most of the calls should be resolved in the Session objects – we introduced a simple fake login in April (you could log as any user using password "guest").

We started to implement the OpenID login support in May (see Section 10.1.4). When we met some technical difficulties, we decided to implement our own registration and login first.

For more information about user management, see part 7.4.

GUI Look and Feel

The design of the GUI was driven by the intentions of effectiveness, while retaining a smooth look at the same time. The effectiveness was achieved by stressing the keyboard-oriented control of the components and simplicity of their layout. The need of a pleasant visual feeling was the reason for using the Twitter Bootstrap design, bringing a user friendly feeling of the application at a very little cost.

GUI Pages

We started having one page only, the Translation Workspace, which was sufficient for development of the application for several months. Later we added the Document Creator page as the

default one, which replaced itself by the Translation Workspace once a document was created, and this was sufficient for some more time. However, we eventually could not avoid adding more pages and the need to explicitly handle multiple pages became obvious.

This led to some problems, because GWT cannot easily handle more pages, which genuinely surprised us.

We describe the problems and their solutions in the chapter [9.3.2](#)

Handling Parsing Errors

We found that since many subtitle files are partly or fully generated by users, and probably also because the SRT format has no official specification, errors in the files are very common. Rejecting even the files with only minor errors (such as a surplus or missing newline, or an incorrect time format which we check thoroughly) showed to be unnecessarily strict, as typically most of the file is correct and the number of errors is small.

We decided to introduce heuristics to decide whether to reject the whole file: if the number of recoverable errors is lower than 10, the erroneous parts are skipped and the rest of the file is parsed.

Offline Mode

At first, our intention was to make the whole application able to run offline. However, it was not a priority and was not even part of the official specification.

When we tried the HTML5 Local Storage, though, the offline mode seemed to be actually easy to implement using this technology, while the support for it is good across web browsers.

The Offline mode is described in bigger detail in Chapter [9.3.5](#)

Subgestboxes

One of the most important features of the application's user interface is the behavior of the actual translation workspace, particularly the text-boxes where user writes their translations and which provide the pop-up suggestions from the translation memory. Since the functional requirements for these boxes were gradually refined throughout the development process, their implementation also underwent several stages of evolution. However, their class name, `SubgestBox`, remained from the early stages, implying the idea of a box offering suggestions for subtitles.

Since the GWT pre-implemented `SuggestBox` did not meet our requirements we had to find our own solution using HTML `IFrames` and GWT `RichTextArea`. It provided all the functionality we have requested, but there was a significant performance problem – the browser often became irresponsive when loading hundreds of them. We solved this by distinguishing so-called “fake” and “real” `SubgestBoxes`; it is described in bigger detail in Chapter [9.3.6](#).

Another issues as focus handling, automatic scrolling, dealing with endlines (see Table [10.1](#) for details) and automatic resizing had to be solved in order to make the Translation Workspace more user-friendly.

Browser	getText()	getHTML()
Firefox	first lines second line	first line second line
Chrome	first line second line	<div>first line</div><div>second line</div><div> </div>
Safari	first line second line	first line<div>second line</div><div> </div>
IE9	first line second line	<p>first line</p><p>second line</p><p> </p>
Opera	first lines second line	<p>first line</p><p>second line</p><p> </p>

Table 10.1: Table capturing different handling of newlines across browsers, showing the return values of the RichTextArea’s `getText()` and `getHTML()` methods. In all the cases, the text entered was “first line[enter pressed]second line[enter pressed]”.

Subtitles Export

When implementing the subtitles export feature, we encountered an unexpected issue: GWT cannot create a new file, neither on server (because it is all JavaScript) nor on the user’s computer (because of security restrictions).

We found several possible solutions:

- export the subtitles into a text area
 - the easiest to implement
 - not what a user expects
 - not easy to use for the user
- send the exported subtitles to user’s e-mail address
 - quite easy to implement because we already have implemented the e-mail sending
 - might be useful for some users
 - not what a user expects
 - can be hard to use for the user
 - might be added as a special feature but should not be the only or primary way
- create a temporary file on the server and offer a download link to the user
 - expected behavior from the user (file download using the browser)
 - reasonably reliable
 - reasonably easy to implement
 - should handle access rights in Jetty which is an added complication
 - creating temporary files is not very clean
- create a new servlet (using JSP) for downloading the file
 - expected behavior from the user (file download using the browser)

- probably the cleanest possible solution
- we were afraid that it would be too much extra work, but we realized that it was not so difficult

Eventually, we decided for the last option as the best one; therefore, the User Space actually consists of two servlets, one to handle RPCs and another for file download.

Chunks saving/loading

We originally decided that the subtitle file will be parsed in the GUI, to avoid unnecessary sending of the data from GUI to User Space and then back from User Space to GUI; parsing the file in User Space would also cause unnecessary load on the server.

Although we wanted to make everything as parallel as possible, we found out that we have to save all the chunks immediately after parsing so that they do not get lost; therefore, requesting the translation suggestions is only done after that, even though this means that the translation suggestions start arriving a little later.

To also enable editing the files, the Translation Workspace became the only page that has two constructors – one receiving the text of a subtitle file to parse, and the other getting an already parsed document from the database.

Finalizing and training the translation pair rankers

Although the candidate search was started early in the development process, the ranking of candidates was not completed until the main development phase. Since we base the ranking on combining various scores with weights, it is necessary to estimate these weights from data. In order to produce the annotated data for this, all other parts of the system had to be fully functional. Because of this, we could finalize the ranking and train the ranking models only later in the main development phase.

Machine translation

Although machine translation with Moses was thought to be just a nice possibility to have and weren't given too much attention initially, at the end we were very surprised with how accurate it is on our data (seeing it being more successful than Google Translate was a *very* gratifying moment) and how quick we managed to finally get it.

The only problematic part was fitting both our application and Moses on the same small virtual machine we were given for this project. However, when we deleted all unnecessary data from the disk and ran with only the binarized models, we were able to run it fine (even when we have only about 1 GB of free disk space, but it seems to be enough for Jetty).

10.1.5 Final Development

We left many issues which we considered to be only minor and technical to the end of the development process; some of these issues were eventually found to constitute intensive work.

We set the Feature Freeze to the 12th August, 12 p.m. After the Feature Freeze, no new features were added to the project, and we started an intensive review of already existing code, debugging, and working on the documentation. After the feature freeze, we improved stability of the application and its responsiveness.

10.2 Evaluating the Development Process

10.2.1 Technologies

One of the crucial decision for the project was the choice of technologies. Most of the technologies we used – Maven, Scala, Hibernate, GWT – were new to some of us and we also had not much experience with the other. Combining these technologies together was a difficult task and would probably even be for an experienced Java developer. Generally, because we were not too very familiar with the technologies, we spent most of the time solving technical issues. There are numerous research challenges, mostly in the fuzzy matching part, which had to remain untouched due to that. Nevertheless, it is doubtful if this was caused only by technical difficulty or by paying too much attention to less important technical tasks.

A bottleneck of the development process was also that not all of us were familiar with all of the technologies. It happened several times that somebody could not continue developing a particular part of the project and had to wait for another team member to fix the issue.

On the other hand, our choice of technologies appeared to also be good since it saved a significant amount of work for us due to the possibility to share an implementation of a class. Using the Scala language for the core also made the parallelization much easier.

10.2.2 Structure

Although we spent a long time on discussions of how the structure of the project would look like, we did not avoid a radical change of the design of the application a few months after the project started. Nearly the whole User Space code had to be dropped and it was also necessary to totally remake the client components existing so far.

10.2.3 Efficiency, communication

During the whole time we had the problem that our development process was not as effective as we would imagine and it was also difficult to find a suitable communication platform. Although we saw each other often to discuss the project issues and had regular meetings, a working online communication played a crucial role for us.

As a very first communication channel, we established a mailing list at Google groups. All the notifications and comments from Github were redirected to that mailing list, as well as results of Jenkins builds. Soon, we started to receive dozens of FilmTit email every day and it started to be quite difficult to follow the conversations and find items in the conversation history.

Because of this, we tried to use PiratenPad². It is free a web-based collaborative real-time editor, allowing authors to simultaneously edit a text document, and see all of the participants' edits in real-time, with the ability to display each author's text in their own color. There is also a chat box in the sidebar to allow meta communication. The service is run by the German Pirate Party. We tried to use the tool to gather personal plans and "todos", bug reports, etc. After some time, the pads became messy and the mailing group became the main communication channel again. We were also solving many issues bilaterally using various instant messaging systems.

In July, we launched a Skype group chat. It effectively replaced the personal meetings, which was much sought after as usually several members were out of Prague at a time, but fortunately, they typically had an Internet connection. It also helped to solve many issues instantly and efficiently. However, it has a same disadvantage as email conversation – it is difficult to easily find facts in the history – but now with hundreds of messages every day to make things worse. Fortunately, because almost every time there were at least some team members online, it was always possible to ask for a summary of the previous discussions and we managed to keep all the team members informed about everything important.

10.3 Work Distribution

Karel Bílek

- implementation of the VLC playback and java applet
- processing the OpenSubtitles.org data
- preparing and optimization of the Moses system to our use
- initialization of Jetty webserver

Josef Čech

- development of the User Space

Joachim Daiber

- design and implementation of the Core Translation Memory, searching, ranking, merging, import, indexing, media source retrieval, etc.
- evaluation of database systems
- parts of the GUI layout, stylesheet and general fixes in GUI
- set up continuous build system, initial Maven project, configuration management

²<http://www.piratenpad.de/>

Jindřich Libovický

- early processing of the opensubtitles.org data
- design and implementation of the User Space

Rudolf Rosa

- development of many functions of GUI (login, Offline Mode, pages, dialogs, settings etc.)
- Remote Procedure Calls and their GUI implementation

Jan Václ

- development of the GUI (structure, visual appearance and experience, Translation Workspace)

10.4 Possible Future Development

There are many issues to be improved, mostly in the graphical interface. It would probably be possible to add new features forever. A newer version would likely include support for more language pairs.

We would like to let the project run for some time and wait if it would find its community of users. If it would reach some success among the user, it could provide much interesting data – mostly about preferences of the users about which translation suggestions are useful for them. Such information could be useful for developing new fuzzy matching techniques including some research challenges, such as finding matches to only a part of a sentence. Success among users would also mean a growing source of well structured data for statistical machine translation.

Part III

Manuals

Chapter 11

Technical Manual

11.1 System Requirements and Setup

11.1.1 Running the Server

The system is cross-platform, requiring only Java 1.6 or newer and Postgres (tested with version 9.1, but should generally work with version >8.3). The system was tested on Windows XP, Windows 7, Mac OS X and Ubuntu.

The system is run from a jar file that contains the full server and all dependencies (the jar file is 108M in size). The server can be run with the following command:

```
java -jar server-0.1.jar configuration.xml 8080
```

Listing 11.1: Running the server

The arguments specify the configuration file (see Section 11.3) and the port the server should run on. Although it is not strictly necessary, it is recommended to run the server with a maximum heap size of at least 500MB (`-Xmx500m`).

11.1.2 Database Configuration

The system will work with any standard Postgres installation, however it is recommended to setup Postgres to use more memory to assure the database runs fast. On our production server (with 4GB of memory), we use the following settings:

```
maintenance_work_mem = 256MB
effective_cache_size = 1500MB
work_mem = 256MB
shared_buffers = 1GB
max_connections = 10
```

Listing 11.2: Production server Postgres settings

To use Postgres with the necessary amount of memory on a Linux system, the kernel settings for shared memory may have to be adjusted, e.g.:

```
# Maximum shared segment size in bytes
kernel.shmmax = 2147483648
# Maximum number of shared memory segments in pages
kernel.shmall = 4194304
```

Listing 11.3: /etc/sysctl.conf on production server

Depending on the system configuration, it may also be necessary to increase `shmpages`. For more information on Postgres memory management and how to adjust the shared memory settings for different operating systems, please see the Postgres documentation on *Resource Consumption*.¹

For full-text search, the full-text search integrated in Postgres is used (this used to be *TSearch2*). By default, a number of languages are available, including English but excluding Czech. If one wants to use additional languages, they must be installed manually. For the Czech language, we prepared an installation script which can run using the following command (an example for running the command on an Ubuntu Linux system).

```
$ ./install_czech_configuration.sh /usr/share/postgresql/9.1/tsearch_data
$ sudo -u postgres ./add_configuration_to_db.sh postgres filmtit
```

Listing 11.4: Installing the Czech text search configuration in a local Postgres database.

11.2 Tasks

11.2.1 Step-by-Step Guide: Setting up the Server with an Existing Database Dump

The following guide contains examples for setting up the system on a plain Ubuntu system.

1. **Copy the contents of the production directory to a local directory.** We use the directory `/production` for illustration. The production directory contains the following files:
 - **server-0.1.jar** – the FilmTit app
 - **configuration.xml** – the default configuration file
 - **models** – text processing models for Czech and English
 - **install_czech_configuration.sh** – helper script for Postgres text search configuration installation

¹<http://www.postgresql.org/docs/9.1/static/runtime-config-resource.html>

- **add_configuration_to_db.sh** – helper script for adding the Czech text search configuration to the database
 - **encz_dump** – dump of the database for the English-Czech language pair
2. **Install and setup a Postgres database.** Before being able to import the database dump to the database, ensure that there is a running version of Postgres.² The system will run with a standard Postgres setup, however it is recommended to adapt the Postgres installation as described in Section 11.1.2.

```
$ sudo apt-get install postgresql
```

Listing 11.5: Installing Postgres on Ubuntu.

3. **Install the Czech Postgres text search configuration:**

```
$ cd /production

#Install the Czech text search configuration in the tsearch_data folder of the Postgres installation
$ ./install_czech_configuration.sh /usr/share/postgresql/9.1/tsearch_data

#Create the empty database
$ createdb filmtit

#Install the text search configuration in the postgres database "filmtit" (user postgres)
$ sudo -u postgres ./add_configuration_to_db.sh postgres filmtit
```

Listing 11.6: Installing the Czech text search configuration to Postgres.

4. **Import the database dump.**

```
$ sudo -u postgres pg_restore -d filmtit encz_dump
```

Listing 11.7: Importing the database dump into the database filmtit.

5. **Ensure the correct JDBC connector is specified in configuration.xml.**
6. (optional) **Ensure the correct Moses server is specified in configuration.xml.** More information on how to install and run the Moses server can be found in in Section 11.5. Currently, we are running a Moses server for testing with our models on server and port `http://u-pl17.ms.mff.cuni.cz:8080`; this is *highly* temporary and *will* be stopped in the near future.
7. **Run the server with the configuration on a suitable port.**

²Postgres installation guide: http://wiki.postgresql.org/wiki/Detailed_installation_guides

```
$ java -jar server-0.1.jar configuration.xml 8080
```

Listing 11.8: Running the server on port 8080.

For the production setting, the server should be run in the background:

```
$ nohup java -jar server-0.1.jar configuration.xml 8080 &
```

Listing 11.9: Running the server on port 8080.

11.2.2 Importing Data

Alignment

For importing the data from the “raw” subtitle files, the directory with .gz files has to be set correctly in configuration.xml in subtitles_folder, the file_mediasource_mapping file has to be set correctly, and the data_folder folder has to exist (also, if there were some previous experiments, the contents have to be removed).

Then, to perform the same alignment as we did (as described in 4.5), the alignment class has to be run as follows:

```
$ java -classpath /deployed/server-0.1.jar cz.filmtit.dataimport.alignment.tasks.FinalAlignment configuration.xml
```

Listing 11.10: Running the alignment

Import

After all necessary paths (most importantly data_folder) are specified in the configuration file, the import can be run as follows:

```
$ java -Xmx3G -classpath /deployed/server-0.1.jar cz.filmtit.dataimport.database.Import configuration.xml
```

Listing 11.11: Running the data import

Setting up the indexes

For the imported translation pairs, indexes for fast retrieval are created by running:

```
$ java -Xmx3G -classpath /deployed/server-0.1.jar cz.filmtit.dataimport.database.Reindex configuration.xml
```

Listing 11.12: Indexing the imported data

11.3 Configuration

The configuration for the server is defined by the file `configuration.xml`, which has to be specified on startup.

In this section, we will give a brief overview over the properties specified in the configuration file and its default values.

11.3.1 General Settings

L1 and L2 specify the ISO 639-1 codes of the source and target languages used in the translation memory.

```
<l1>en</l1>
<l2>cs</l2>
```

Listing 11.13: Languages

11.3.2 Database

The database connection must be specified as a valid JDBC connector. By default, the DBMS is the local Postgres database `filmtit` with default username and password.

```
<database>
  <connector>jdbc:postgresql://localhost/filmtit</connector>
  <user>postgres</user>
  <password>postgres</password>
</database>
```

Listing 11.14: Database connection

11.3.3 Text Processing Models

For various text processing tasks within the translation memory, it is necessary to specify a number of model files.

The system will search for the models in the folder `model_path`.

```
<model_path>models</model_path>
```

Listing 11.15: Model path

OpenNLP Maximum Entropy tokenizer models are specified in the `tokenizers` section. If for a specific language no tokenizer model is specified, the translation memory will use the default OpenNLP `WhitespaceTokenizer`.

```
<tokenizers>
  <tokenizer language="en">en/token.bin</tokenizer>
  <tokenizer language="cs">cs/token.bin</tokenizer>
</tokenizers>
```

Listing 11.16: Tokenizers

OpenNLP Maximum Entropy models for Named Entity Recognition are specified in the `ner_models` section. Each `ner_model` specifies a language (ISO 639-1 code) and the type of Entity that it recognizes. Currently, only Person, Place and Organization are used. If fewer models are specified, only the specified models will be used.

```
<ner_models>
  <!-- English -->
  <ner_model language="en" type="Person">en/ner-person.bin</ner_model>
  <ner_model language="en" type="Place">en/ner-place.bin</ner_model>
  <ner_model language="en" type="Organization">en/ner-organization.bin</ner_model>

  <!-- Czech -->
  <ner_model language="cs" type="Person">cs/ner-person.bin</ner_model>
  <ner_model language="cs" type="Place">cs/ner-place.bin</ner_model>
  <ner_model language="cs" type="Organization">cs/ner-organization.bin</ner_model>
</ner_models>
```

Listing 11.17: Models for Named Entity Recognition

11.3.4 Data Import

For the data import as described in Section 11.2.2, several files have to be specified.

- `subtitles_folder` – the folder containing the subtitle files for the initial import
- `data_folder` – the folder for the results of the alignment (see section 11.2.2)
- `file_mediasource_mapping` – a CSV file that describes the sources (movie or TV show) of the subtitle files
- `batch_size` – the number of subtitle files that should be processed at the same time; a higher number will increase the memory consumption of the import process
- `mediasource_cache` – the location of a cache file for the movie data queried from an external API for each subtitle file

```
<import>

  <subtitles_folder>/filmtit/data/export/files/</subtitles_folder>

  <data_folder>/filmtit/data/aligned/</data_folder>
  <file_mediasource_mapping>/filmtit/data/files/export_final.txt</file_mediasource_mapping>
  <batch_size>100</batch_size>
  <mediasource_cache>/filmtit/data/imdb_cache</mediasource_cache>

</import>
```

Listing 11.18: Settings for the Data Import

11.3.5 Module-Specific Options

Core TM

The module-specific options for the Core TM are mostly related to performance.

- `max_number_of_concurrent_searchers` – specifies the maximum number of searchers that will be created concurrently. By default, 5 searchers will be created and requests will be scheduled among them.
- `searcher_timeout` – specifies the maximum time the scheduler will wait for a searcher to respond. If the time is exceeded, the scheduler will retry with a different searcher.
- `ranking` – specifies models used for different rankers; the model files are serialized WEKA classifiers.

```
<core>
  <ranking>
    <exact_ranker_model>ranking/exact.model</exact_ranker_model>
    <fuzzy_ranker_model>ranking/fuzzy.model</fuzzy_ranker_model>
  </ranking>

  <max_number_of_concurrent_searchers>5</max_number_of_concurrent_searchers>
  <searcher_timeout>60</searcher_timeout> <!--in seconds-->
</core>
```

Listing 11.19: Settings for the Core TM

User Space

The User Space settings contain constants influencing the behavior of the application and also fields that need to be set correctly to make the application run.

- `maximum_suggestions_count` – default maximum number of translation suggestions which are displayed to the user for one chunk; users can change this value in their user settings
- `session_timeout_limit` – time (in milliseconds) of user inactivity after which the user session is terminated (in case the user did not turn on the permanent login)
- `permanent_timeout_limit` – time (in milliseconds) of user inactivity after which the user session is terminated (in case the user did turn on the permanent login)
- `server_address` – URL of the web application, used especially in OpenID login to provide the OpenID provider with a return URL to redirect the user to after authentication

```
<userspace>
  <maximum_suggestions_count>5</maximum_suggestions_count>
  <session_timeout_limit>3600000</session_timeout_limit>
  <permanent_session_timeout_limit>1209600000</permanent_session_timeout_limit>
```

```
<server_address>http://localhost:8080</server_address>
```

Listing 11.20: First part of the User Space settings

The second part of the User Space setting concerns sending email from the application. The administrator can set the outgoing email server, account details and texts of the messages. Settings needed to establish the connection are listed first.

- `mail.transport.protocol` – protocol for sending
- `mail.stmps.port` – port for sending mail
- `mail.stmps.host` – host server
- `mail.smtps.socketFactory.class` – class used for ssl encryption
- `mail.smtps.socketFactory.port` – port used for ssl
- `mail.smtps.auth` – if connection needs authentication
- `mail.filmtit.address` – account login
- `mail.filmtit.password` – account password

Settings of the email subjects and bodies follows. In the registration email body you can use `%userlogin%` and `%userpass%` templates which are later substituted by the real user name and password values at the time the email is sent. Similarly, `%userlogin%` and `%changeurl%` templates can be used in the email which is sent to the user when they forget their password.

- `mail.filmtit.forgottenPassSubject` – email subject for forgotten password email
- `mail.filmtit.forgottenPassBody` – email body for forgotten password email
- `mail.filmtit.registrationSubject` – email subject for registration email
- `mail.filmtit.registrationBody` – body of email which is sent during classical registration
- `mail.filmtit.registrationOpenIDBody` – body of email which is sent during openID registration

```
<mail>
  <properties>
    <comment/>
    <entry key="mail.transport.protocol">smtps</entry>
    <entry key="mail.smtps.port">465</entry>
    <entry key="mail.smtps.host">smtp.gmail.com</entry>
    <entry key="mail.smtps.starttls.enable">true</entry>
    <entry key="mail.smtps.socketFactory.class">javax.net.ssl.SSLSocketFactory</entry>
    <entry key="mail.smtps.socketFactory.port">465</entry>
    <entry key="mail.smtps.auth">true</entry>
    <entry key="mail.filmtit.registrationSubject">Registration on Filmtit</entry>
```

```

<entry key="mail.filmtit.forgottenPassSubject">Request for changing your password on Filmtit</entry>
<entry key="mail.filmtit.registrationOpenIDBody" >
    Thank you for using FilmTit. Although you are using OpenID,
    we have created a FilmTit registration for you in
    case that there should be any problems with your OpenID
    provider. This is just a backup account for you to
    keep and you can safely continue using your OpenID login.
    Your login: %userlogin%
    Your password: %userpass%
</entry>
<entry key="mail.filmtit.registrationBody">
    Hello %userlogin%,

    Thank you for using FilmTit.
    Your registration was successful and you can start translating now!
    See you at http://filmtit.cz

    FilmTit
</entry>
<entry key="mail.filmtit.forgottenPassBody">
    Hello,
    there was a request for changing the password
    for the account: %userlogin%
    You can change your password by clicking on this link: %changeurl%
</entry>
<entry key="mail.filmtit.address">filmtit@gmail.com</entry>
<entry key="mail.filmtit.password">jkhrjj2012</entry>
</properties>
</mail>

```

Listing 11.21: Settings for sending emails, second part of the User Space settings.

The last part of the User Space configuration is configuration of OpenID for Seznam.cz.

- `seznamcz` – URL of the Seznam.cz OpenID endpoint
- `nickname` – name of the URL variable with the authentication data

```

<openid>
  <seznamcz>http://id.seznam.cz/yadis</seznamcz>
  <nickname>sreg</nickname>
</openid>
</userspace>

```

Listing 11.22: First part of the User Space settings

APIs and Keys

The configuration also specifies the URL of the running Moses instance and the keys for external APIs. A Freebase key can be obtained via the Google API Console.³

³http://wiki.freebase.com/wiki/How_to_obtain_an_API_key

```
<mosesURL>u-pl17.ms.mff.cuni.cz:8080</mosesURL>  
<freebase_key>AlzaSyCBD3hth3xIXTa9FDet4zMtAh0vAjtvp0</freebase_key>
```

Listing 11.23: APIs and keys

11.4 Adapting the TM to new Language Pairs

The implementation currently uses the English-Czech language pair, however, the system is easily adaptable to new language pairs. In this section, we are briefly describing the necessary steps to adapt the system to a new language pair.

11.4.1 Basic Setup: Exact and Fuzzy Matching

The most basic setup for internationalization uses the following backoff translation memories:

- database-based exact retrieval and ranking
- database-based fuzzy retrieval and ranking

Both levels work out of the box for the following languages that have a Postgres text search configuration by default:

- Danish
- Dutch
- English
- Finnish
- French
- German
- Hungarian
- Italian
- Norwegian
- Portuguese
- Romanian
- Russian
- Spanish

- Swedish
- Turkish
- default “Simple” configuration

Additional text search configurations can also be found online.

Necessary steps for setting up a new language pair

1. Specify the language pair in `configuration.xml`. If the language does not belong to the languages above, it has to be added to the class `cz.filmtit.share.Language`.
2. Currently, only sentence tokenizers for Czech and English are included, which means that for additional languages, a sentence tokenizer like e.g. `EnglishSentenceTokenizer` that extends `SentenceTokenizer`⁴ has to be created.

After these steps have been taken, the data has to be aligned, imported and indexed. For more information, see Section 11.2.2.

11.4.2 Advanced Setup: Program-based Signatures and Machine Translation

A more advanced setup can use the levels from the basic setup, or other program-based signature levels, as well as machine translation.

For this, it is necessary to do tokenization using an OpenNLP tokenizer. Although it is possible to do this with a standard (unsupervised) tokenizer, it is recommended to use a tokenizer specifically trained for a given language. Pre-trained models for some languages are available on the OpenNLP website⁵ and on github.⁶ The training of a tokenizer for a new language is described in Section 5.3.2 on the example of Czech.

For instructions on how to set up Moses as a backoff level, see Section 5.3.8.

11.5 Running the Moses Server

The Moses server can be run either on a separate computer, or on the same computer as the FilmTit system. The main process connects to the Moses server via HTTP and requests sentence translations through remote procedure calls. However, there is no authentication, so the Moses server has to be secured in other ways, for example by the network infrastructure; if it is not, there is a risk of unauthorized translation requests.

Moses is an experimental software, which is one of the reasons for its slightly complicated installation. However, we will present an easy way to install Moses on an operating systems with the APT package system (Ubuntu, Debian) and a way to run Moses with our models, which can

⁴In the package `cz.filmtit.share.tokenizers`.

⁵<http://opennlp.sourceforge.net/models.html>

⁶<https://github.com/utcompling/OpenNLP-Models>

be found on the accompanying portable media; on different operating systems, the steps should be similar, but the official Moses website⁷ should be consulted for further information.

Except for APT, git is needed for the initial download and GCC for building of Moses. If libboost and libxmlrpc are not installed, sudo privileges are also required.

11.5.1 Installing the Moses Server

1. First, Moses has to be checked out from the official git repository.

```
git clone git://github.com/moses-smt/mosesdecoder.git
```

Listing 11.24: Checking out the Moses from repository

2. Second, the boost libraries have to be installed. If they are not installed, it can be done by the following command.

```
sudo apt-get install libboost-all-dev
```

Listing 11.25: Installing boost from APT

3. An XML RPC package has to be installed, too. If it is not, this can be corrected by the following command.

```
sudo apt-get install libxmlrpc-c3-dev
```

Listing 11.26: Installing XML RPC from APT

4. Now, the Moses server can be installed. If, for example, 4 cores are available, Moses can be built as follows.

```
cd mosesdecoder
./bjam --with-xmlrpc-c=/usr/bin/ -j4
```

Listing 11.27: Building Moses

If boost has been installed to a non-standard location, it has to be specified with the `--with-boost=/path/to/boost` switch.

5. The Moses server should now be compiled at `mosesdecoder/bin/mosesserver`.

11.5.2 Running the Moses Server

For running the Moses server with our data, the folder with our models is needed. This data is supplied with the project.

Now, let us suppose that our data with models are at folder `/data/` and `mosesdecoder` is at folder `/bin/`. Then, the server can be easily started at port 8081 with 4 threads (the recommended number of threads is the number of cores) by running

⁷<http://statmt.org/moses>

```
cd /data/  
/bin/mosesdecoder/bin/mosesserver -threads 4 --server-port 8081 -f config.ini
```

Listing 11.28: Running Moses server

and letting the Moses server start. Nothing else is needed.

Chapter 12

User Manual

12.1 About the FilmTit Application

FilmTit is a web application that assists amateur subtitle translators with translating movie and TV shows subtitles. In order to help save the amount of work spent on the translation, it provides suggestions on how the subtitles could be translated, based on a database of already existing translated subtitles.

You can translate any subtitle file you have from English to Czech or from Czech to English, making use of the millions of translations already made by other movie subtitles translators, coming from the tens of thousands of subtitle files in our database. From these, we always carefully select the most relevant ones for the lines that you are translating at the moment, which you can use as they are, post-edit them a little, or just use them for inspiration. And even if we find no similar lines in our database, there is always the machine translation system, ready to provide an automatically generated translation for any line you encounter. (Currently, the machine translation is only available for translation from English to Czech. Also please note that the machine translation can contain mistakes, as there is no perfect machine translation system in existence.)

Often it may be hard to fully understand the subtitles without seeing the movie; therefore, you have the possibility to load a movie file into the application, and the part of the movie that you are translating at a given moment will always be played to you, also showing the source subtitles (and the target ones as well if you have already translated them). Most of the movie formats and codecs are supported.

FilmTit is a web application, which means you just have to open your favourite internet browser and you can start translating straight away – no installation is necessary (although you may have to install Java and VLC for the video playback function, see Section 12.2). However, this does not mean that you have to be online to translate the subtitles! Once you get all the translation suggestions from the server, you can go to the Offline Mode, work on the translation offline, and all your work will be automatically saved on the server when you are online again! See Section 12.7.

Similarly to Google Documents, you do not have to worry about saving your work – your translations are automatically saved online right after you type them!

To run the application, you need a web browser with HTML5 support (Opera v. 12 Firefox v. 14, Chrome v. 21, Safari v. 5.1.5, or higher). To use the optional video playback in your browser, you also need to have Java (at least version 1.6) and the VLC plugin (at least version 1.1.4).

12.2 Installing Java and VLC Plugin

Having installed Java and VLC Plugin is necessary for using the video playback in the application. It still of course possible to use the application without these plugins if you are not planning to use the video playback.

12.2.1 Installing Java

If you do not have Java installed, your browser will notice that automatically and will suggest you to install the missing plugin. If that happens, please follow the browser instructions.

If the installation via the browser fails, you will need to install Java manually, which is described in the following paragraphs.

Windows

For a manual installation, first download the Java Installer from the *java.com* website. Go to <http://www.java.com> and click on “Free Java Download” button. The website will propose a suitable version for you. (If you want to download a different version, click on the “See all Java downloads” link.) It is recommended to check the system requirements of the version you are going to download and to read the license conditions.

After launching the installation guide, simply follow the instructions displayed. For a more detailed description of the installation, please see http://www.java.com/en/download/help/windows_manual_download.xml.

After finishing the installation, it is necessary to restart your browser. (It is recommended to reboot the whole system.)

Linux

To install Java on a Linux system, download the installation package similarly as is described for Windows. To do the installation from the command line, please follow the instructions at http://www.java.com/en/download/help/linux_install.xml#install. (It might not be enough to just install Java – it may also be necessary to manually enable Java in your browser. The instructions can be found on the same website.)

After finishing the installation, it is necessary to restart your browser.

12.2.2 Installing VLC Plugin

The VLC player plugin is available only for Firefox, Chrome, Opera and Safari. Internet Explorer uses an *ActiveX* VLC plugin which is not supported by our application; therefore, video playback

unfortunately does not work in Internet Explorer.

If you already have an installation of the VLC player without the plugin, it is necessary to reinstall the whole VLC player. It is not necessary to uninstall it manually, it is done automatically with the new installation.

To install the VLC player with the plugin on Windows, download the installation program from <http://www.videolan.org/vlc/>. The web page should suggest you a suitable version for your computer. If you want to install a different version, click on “Other Systems and Versions”, otherwise just click the “Download VLC” button.

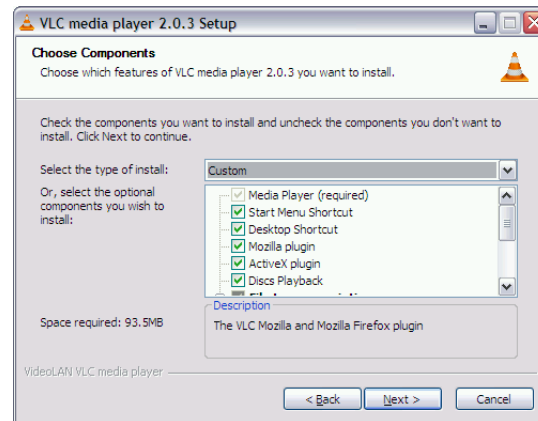


Figure 12.1: Installation guide of VLC player for Windows.

To install the plugin, do not forget to tick the box with “Mozilla plugin” label in the third step of the installation guide (named Choose Components, see Figure 12.1). After finishing the installation, it is necessary to restart your browser.

Instructions for installation on Linux systems can be found at <http://www.videolan.org/doc/vlc-user-guide/en/ch03.html>

12.3 Registration and Login

We require the users to be logged into the application during their work. We do so to enable the users to save their work and return to it another time. However, there is also an Offline Mode, where the data is stored locally in your computer and uploaded to server once you go online – see Section 12.7.

12.3.1 Registration and Basic Login

The first option how to get an account to the application is to register and get a user name and password, similarly to any other web application. However, if you have a Google, Yahoo or Seznam account, we recommend you to use the “OpenID login”, which enables you to use your already existing account at Google, Yahoo or Seznam to log into the FilmTit application.

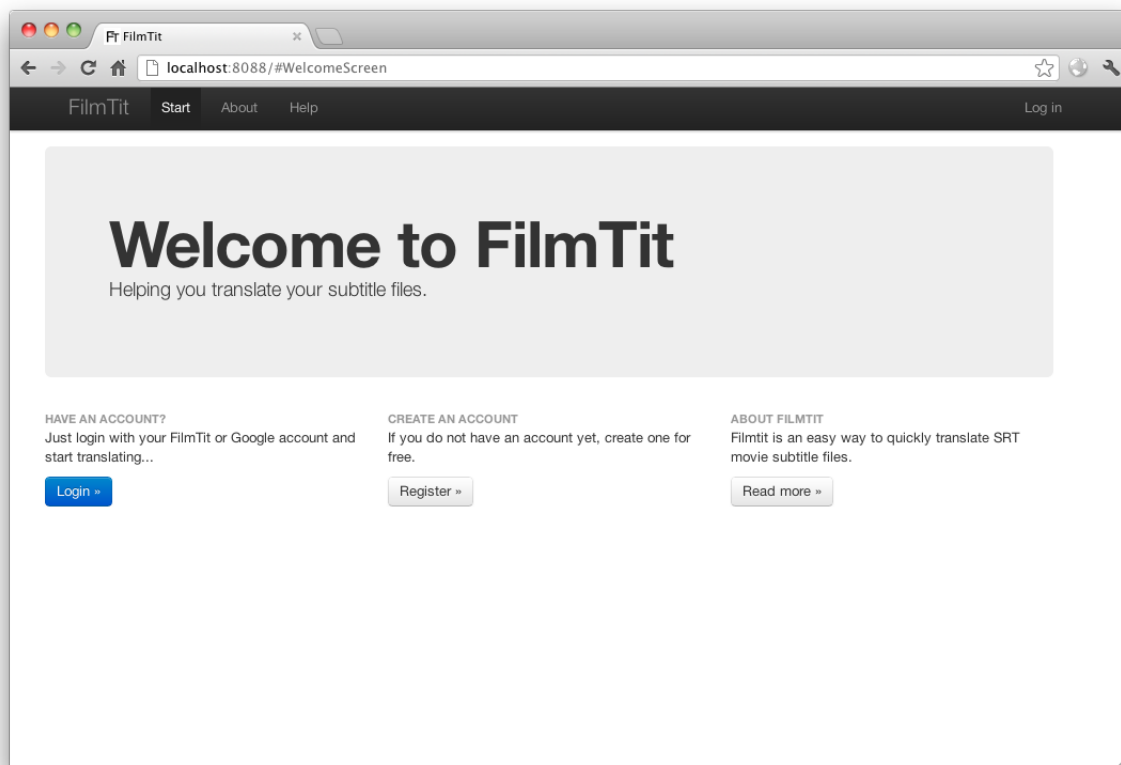


Figure 12.2: Welcome screen of the application.

For the classic registration, click the “Register” button on the Welcome screen (Figure 12.2) or click the “Login” button and select the third tab in the opened dialog (Figure 12.3).

After that, you are requested to choose a user name, fill in a valid email address and type twice the password you would like to use. (You do not need to fill in an email address, but it is necessary for password recovery in case you forget your password.) Because the application does not contain any sensitive information, we try to keep the registration and login process as simple as possible and there are no requirements on the strength of the password (except for a minimum length of 3 characters). After a successful registration, you will receive an email confirming the registration.

You are automatically logged in after the registration. For logging in next time, click on the login button on the welcome screen and fill in your user name and password (Figure 12.3). Your login session is valid for 1 hour – if you do not use the application for 1 hour, you will be logged out automatically. If you want to stay logged in permanently, you can set this in the settings, see Section 12.4.

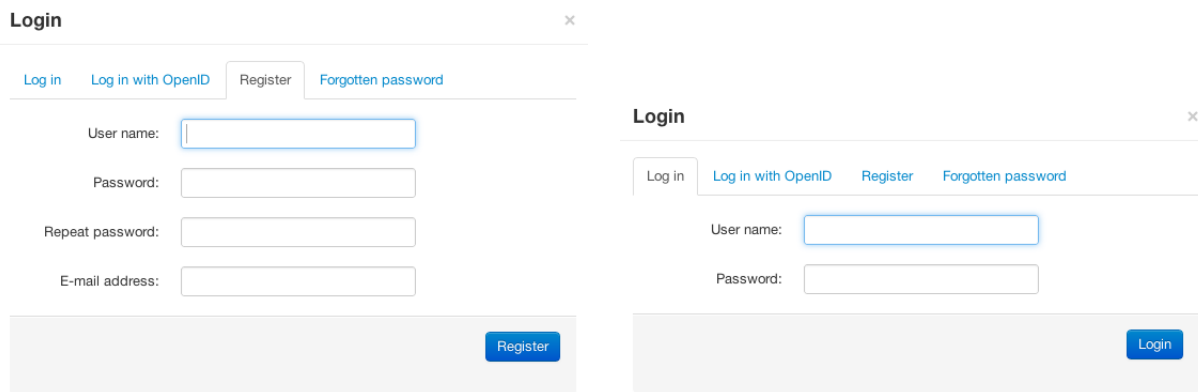


Figure 12.3: Registration form and login form.

12.3.2 OpenID Login

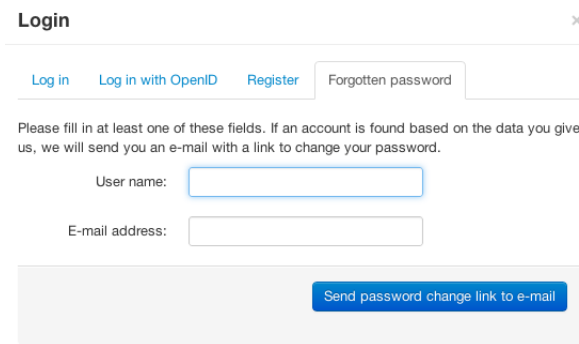
Another option to log into the application is using Google, Yahoo or Seznam account. After choosing the service you want to use, a pop-up window is opened. It may happen that your browser blocks this window – if this happens, you need to allow the pop-up window to continue the logging process.

You will see the login form of the service you have chosen. If you are currently already logged into the service, you will only see the confirmation request to allow the FilmTit application to access your account data. It is your user name in the service, your first name, surname, email and gender, depending on what you filled in in the particular service and what you allowed to be provided to the third party applications. Nevertheless, it is never the password to the original service. FilmTit does not use or keep any of the information provided by OpenID, except your email address. After submitting your user name and user password and confirming that the FilmTit application can receive your authentication data, the pop-up window will be automatically closed. Within a few seconds you will be redirected to the list of documents you own (if this is your first time logging into the application, you have no documents yet, so you will see the New Document page instead).

If you use OpenID Login, you do not have to register – you are registered automatically on your first successful login. You also automatically get a user name and password for the Basic Login, which is sent to your e-mail address upon registration (if your OpenID provider provides us with one) and can be changed in the Settings.

12.3.3 Forgotten Password

Another issue connected to login is dealing with the situation when users forget their passwords. If such a situation happens, open the login dialog and click on the "Forgotten password" tab. Fill in either your user name or email (or both) and click "Send password change link to email" (see Figure 12.4). (Please note that if you did not set a valid e-mail address with your account, you cannot use this feature.)



The screenshot shows a web form titled "Login" with a close button (X) in the top right corner. Below the title, there are four links: "Log in", "Log in with OpenID", "Register", and "Forgotten password". The "Forgotten password" link is highlighted with a blue border. Below the links, there is a paragraph of text: "Please fill in at least one of these fields. If an account is found based on the data you give us, we will send you an e-mail with a link to change your password." Below this text, there are two input fields: "User name:" and "E-mail address:". Both fields are empty. Below the input fields, there is a blue button with the text "Send password change link to e-mail".

Figure 12.4: Form for requesting the forgotten password.

After that, you will receive an email containing your user name with a link to a page where you can change your password. If you ignore the email, the original password will remain valid.

12.4 Changing the User's Settings

You can change the user settings by clicking the Settings link in the top menu of the application (you must be logged in to have the Settings available). You can see the settings form in Figure 12.5. After you are done with changing the settings, click the Save button.

12.4.1 Account and Logging in

User name

The user name has to be unique in the FilmTit application. If users use the classic registration form, they can choose their own user name. If a user wants to register a user name which already exists, the application displays warning. Registration by openID is the second way how to receive a user name. The user name from this registration is extracted from the email address. There is a chance that two users have a very similar email address and the extracted name will be the same. Our app generates the user name with a unique numeric code in this case. The user name can be changed in the page User Settings.

New password

You can change your password by filling the two boxes with two identical strings which will become your new password. As was already mentioned, we do not have any requirements on the strength of the password except for a minimal length of 3 characters.

By leaving the two input boxes empty, the old password remains unchanged.

User Settings sample_user

Account and Logging in

User name:

New password:

Repeat new password:

E-mail address:

Stay logged in: ☐

Translation Workspace

Maximum number of suggestions to show for each line:

Include machine translation: ☒

Figure 12.5: The settings form

E-mail address

In this input box, you can change your email address. It is checked whether the address has a valid email address format, but we do not test the email address' existence and functionality. We recommend to fill in a working email address for the case that you forget your password.

Stay logged in

By ticking this option you stay permanently logged in to the application – unless you log out. (After a really long time of not appearing in the application, you will be automatically logged out for security reasons; it is a month by default, but it depends on the administration settings of the server.)

12.4.2 Translation Workspace

There are also some options concerning the translation workspace. To fully understand the options, please read Section 12.6 first.

Maximum number of suggestions to show for each line

It is the maximum number of suggestion that can be displayed for a particular subtitle chunk being translated. It can be any number between 1 and 100. To work efficiently with the translation suggestion, we recommend to use at most 25 suggestions.

Include machine translation

By this option you indicate if you want to include automatic translation among the translation results. If this option is disabled, you receive only sentences which have occurred before in the subtitle files that we have available in our database.

The machine translation provides automatically generated sentences by the open-source statistical machine translation system Moses. When we tested it on the subtitle data, it performed better than the popular Google Translate system (tested in August 2012).

If you disable the machine translation, you often do not receive any suggestions for chunks. However, you get the suggestions faster (the machine translation is usually the slowest part of the suggestions generation process), and all suggestions you get are human translations which generally have a higher quality than the automatic translations.

Please note that in the current version, machine translation is only available for translations from English to Czech.

12.5 Creating a New Document

A document is a subtitle file in the source language (usually English) which you load into the application, together with its translation in the target language (usually Czech) which you produce with the help of the application. Creating a document means loading a subtitle file in the source language and starting to translate it. You can create a new document either by clicking the “Create a new document” button in the document list, or by clicking the “New document” link in the top menu.

While creating a document, you are asked to fill in the movie title and the document title (which defaults to the movie title, but you can set any name you like). In the case of TV series, please fill in the name of the series, not the name of the particular episode. An example of it can be to type “Lost” as the movie title and “Lost S01E01” as the document title. Then you are asked to choose the source language of the subtitles, encoding of the subtitle file and the path to the actual subtitle file you would like to translate. The only supported subtitle file format is SRT, a simple text format containing the subtitles and their timings (which are real times, in contrast to e.g. SUB format where the timing is given as a number of frames). You should also make sure you selected the proper encoding of the source subtitle file. You can choose from UTF-8, windows-1250 and iso-8859-2, which are the most commonly used encodings for Central European languages. (Usually UTF-8 is the correct choice.)

There is also an option to play the video of the movie you have on your computer. If you want to do so, click the “Load” button below the “Movie playback” headline and select the movie file.

New Document Start a new subtitle document

Title of the movie or TV show:

Document title:

Translation direction:

Load subtitles file

Choose file encoding: ☒ UTF-8
☐ windows-1250
☐ iso-8859-2

Choose file with subtitles: District 9 - en.srt

Movie playback

Read this before using the playback

Movie address on the disk:

Figure 12.6: Form for creating a new document.

For this step you need to have Java and the VLC plugin installed as was mentioned before. Please be patient while doing it, loading the open file dialog can take a while on slower computers.

Then you can submit the document. Within seconds, a form containing movies with the title you provided should appear (see Figure 12.7).

After clicking on the movie you meant, click submit and you can start editing your new document. In case you do not like the suggested movies at all, you can click the cross in the top right corner of the form and try to reset the movie later in the document list.

12.6 Document Editing

When you start editing a document, either a new one or an existing one, you see the translation workspace, see Figure 12.8. It has three columns. In the first column, there are the timings of the subtitles, in the middle column you can see the subtitles in the original language and in the third column there are the text boxes ready to be filled in by the translation in the target language.

Immediately after you open the translation workspace the translation suggestions starts to be loaded.

After the translation suggestions arrive to the translation workspace, you can write down your translations. (You can edit it even if the suggestion does not arrive, but will not be able to see the suggestions.) The translation suggestions appear below the text area where the text cursor is in. You can select one of the suggestions by clicking on them or using the arrow keys and post edit it then. You can also write the translation from scratch and ignore the suggestion. You can add a line break by pressing *Enter*.

To move to the next subtitle chunk just click to the next text box or press the *Tab* key. If you want to move to the previous subtitle chunk, press *Shift + Tab*.

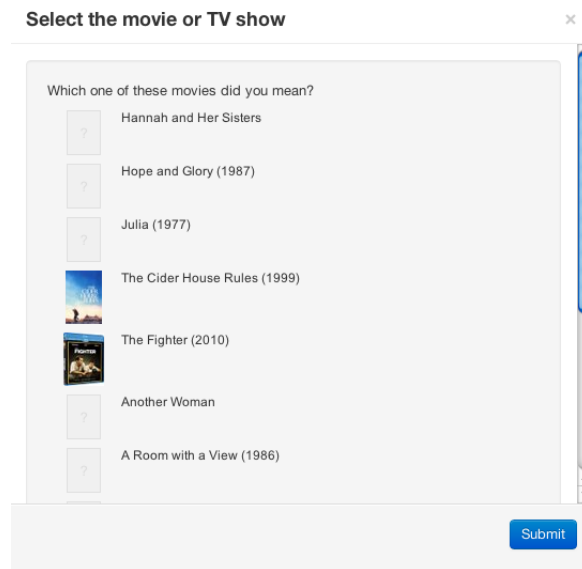
A screenshot of a web application window titled "Select the movie or TV show". Inside the window, there is a text input field at the top. Below it, a message says "Which one of these movies did you mean?". A list of movie titles is displayed, each preceded by a small square placeholder icon. The titles are: "Hannah and Her Sisters", "Hope and Glory (1987)", "Julia (1977)", "The Cider House Rules (1999)", "The Fighter (2010)", "Another Woman", and "A Room with a View (1986)". The first four titles have placeholder icons that appear to be question marks, while the last three have actual movie posters. A blue "Submit" button is located at the bottom right of the window.

Figure 12.7: Form for selection of a movie. It shows the suggestion after a misspelled title of Woody Alan’s movie "Hannah and Her Sisters" was submitted.

You can change the subtitle timing by double-clicking on it or the text of the original subtitle also by double-clicking on it. If you change the text for the particular subtitle, the suggestions are regenerated. It may take some time to the new suggestions to appear.

It is not necessary to save your work in any way, everything is save right after it is edited, so you can leave the document by clicking on a link or even close the browser and nothing will be lost. If the Internet connection breaks down you can continue working on the in the Offline Mode which is described in the following section.

12.7 Offline Mode

When the application realizes that it cannot connect to the server, it offers you to continue in Offline Mode (see Figure 12.9). When you turn on the Offline Mode, you can continue translating the document – all translation suggestions that have already been loaded will be shown to you and all translations that you enter will be saved. (However, you cannot list the documents, open or create another document, or export or delete the documents.)

If you refuse starting the Offline Mode, all the editing done without the Internet connection will be lost!

During the work in Offline Mode, all the operations are stored in the browser. Once you leave the page with the translation workspace, you cannot continue editing the file – however, you can even close your browser or restart the computer, and still all the changes that you have made on the document in Offline Mode will be saved.

The data from the Offline Mode are posted to the server at the time you log in to the application the next time after you confirm you want to do so (see Figure 12.10). If your Internet

FilmTit

About

Document list

New document

Settings

Log out user sample_user

00:01:57,377 - 00:02:01,172	God, she's beautiful.	Bože, ona je nádherná.
00:01:57,377 - 00:02:01,172	She's got the prettiest eyes.	Má ty nejkrásnější oči.
00:02:01,297 - 00:02:03,758	She looks so sexy in that sweater.	Vypadá tak sexy v tom svetru.
00:02:04,718 - 00:02:08,054	I just want to be alone with her and hold her and kiss her	V tom svetru je tak sexy.
00:02:08,179 - 00:02:12,309	and tell her how much I love her and take care of her.	Chci s ní být o samotě objímat ji a líbat ji,
00:02:12,434 - 00:02:15,312	Stop it, you idiot!	Zastav, ty idiote!
00:02:12,434 - 00:02:15,312	She's your wife's sister.	Zastav, ty idiote! Stop it, you idiot!
00:02:15,437 - 00:02:20,734	I can't help it.	Přestaňte, vy idioti! Stop it, you idiots!
00:02:15,437 - 00:02:20,734	I'm consumed by her.	Přestaň, ty idiote! Stop it, you idiot!
00:02:15,437 - 00:02:20,734	It's been months now.	Nechte toho, vy idiote. Stop that, you idiot!
00:02:20,859 - 00:02:24,362	I dream about her, I think about her at the office.	Zastav ten autobus, tupče! Stop the bus, you idiot!
00:02:25,405 - 00:02:28,283	Oh, Lee.	Ne, zastav... ty pitomče! No, stop... you idiot!
00:02:25,405 - 00:02:28,283	What am I gonna do?	
00:02:29,409 - 00:02:33,330	I hear myself mooning over you, and it's disgusting.	

Figure 12.8: The translation workspace during translating a document.

connection starts to work during your work in Offline Mode, you can just reload the page in your browser, log in and continue with your work.

The information in Offline Mode are bound to your computer, browser and user. So, to be able to upload the Offline changes to the server, you have to log in on the same computer as the same user and use the same browser.

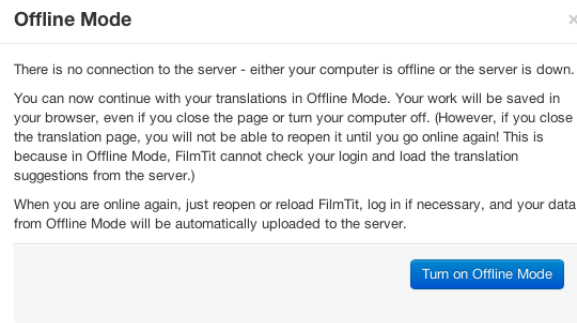


Figure 12.9: Confirmation request for start the offline mode.

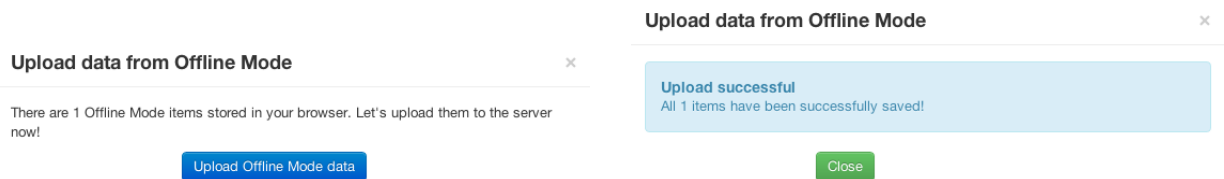


Figure 12.10: Loading data from the Offline Mode

12.8 Operations with Documents

You can list your documents by clicking on the "Document list" link in the top navigation of the application, see Figure 12.11. You can edit the document by clicking on the "Edit" button.

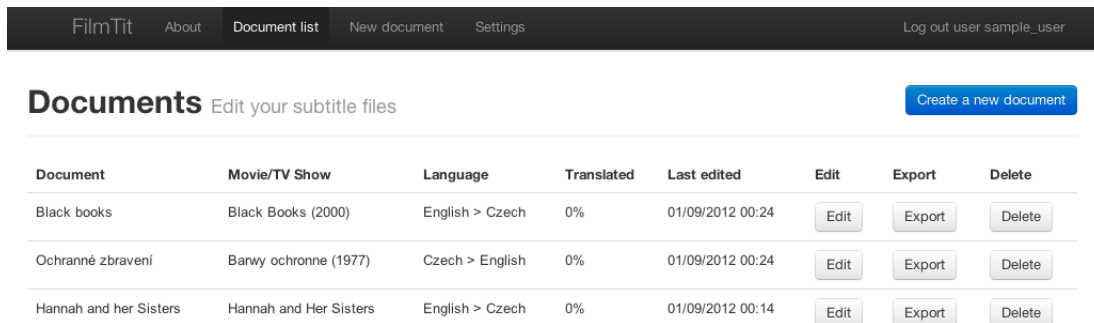


Figure 12.11: List of documents owned by the user

Clicking on the export button will open a dialog for downloading the subtitle file based on the document. You can select if you want to download the subtitles in the source language, the translated version or the translated version with the original subtitles where the document remained untranslated. After clicking on a button with the required format, the download of the

subtitle file will start.

By clicking the delete button you will remove the document from you document list.

You can also change the title of the document, by clicking on the title and typing the original or change the movie title. If change change the name of movie, the same dialog as while creating a document will show (Figure 12.7) where you can select the movie you meant.