Daemonizing and enhancing IceNLP for the purpose of machine translation

Independent study
Reykjavík University

Hlynur Sigurþórsson

Fall, 2010

Contents

1 Introduction

This report describes my independent study on daemonizing and enhancing IceNLP (Natural Language Processing Toolkit for Icelandic) for the purpose of using it in Machine Translation (MT). The motivation for this study was to use the outcome in an ongoing project funded by the Rannís Grant of Excellence, "Viable Language Technology beyond English - Icelandic as a test case", where one of the work packages consists of developing a rule-based MT system from Icelandic to English (hereafter refereed to as IS-EN) using IceNLP and the Apertium machine translation platform. During the development of the above mentioned project the researchers encountered number of problems when integrating IceNLP into Apertium. We have summarized these problems into two categories:

1. *Load delay*: Many of the modules in IceNLP use number of large lexicons in their processing. These lexicons are loaded into memory before processing starts and are kept there until the processing terminates. Using these modules unmodified with Apertium would add intolerable delay to the translation process where the lexicons would be reloaded each time the translation system is used.

2. *Output formation*: The output from some IceNLP modules was not compatible with Apertium. For example, morphosyntactic tags produced by the Part-Of-Speech (POS) tagger IceTagger[1] needed to be mapped to the tagset used by Apertium and some Icelandic multiword expression needed special handling for Apertium.

In order to use modules from IceNLP in the Apertium translation pipeline, various enhancements were needed. First, we added a client-server functionality into IceNLP, where the server stores all necessary lexicons in memory for reuse. Second, we added a Mapping lexicon with various output mapping and formation rules that allow users to modify the output, both the formation and some part of the results.

This report is structured as follows. In section 2 we will discuss the Apertium translation platform, what it contains and how it operates and in section 3 we will discuss the IceNLP toolkit. Then, in section 4, we will discuss the integration of IceNLP and Apertium and the modification we did on IceNLP. In section 5 we will discuss the scalability of the IS-EN system and finish by discussing a website and a webservice that were developed for making our translation system available for on-line demonstration.

2 Apertium

Apertium[2] is a language independent Shallow-Transfer MT platform [9]. It was initially designed for translation between related language pairs but has also been adapted for other language pairs as well [10]. The whole platform is open-source, both programs and data are licensed under the Free Software Foundation's General Public Licence[3] (GPL) and all the software and data for

[1]IceTagger is part of the IceNLP toolkit.
[2]http://www.apertium.org/
[3]http://www.fsf.org/licensing/licenses/gpl.html

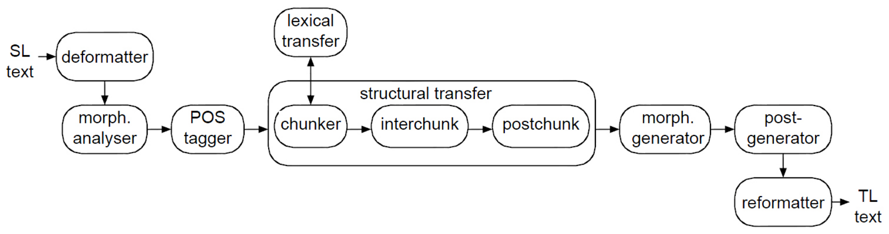


Figure 1: The modular architecture of the Apertium platform.

the 23 supported language pairs[4] (including additional pairs under development) are available in the Apertium repository.

The structure of Apertium has been described as a "Good-old 70's Unix style" [9]. This description is derived from the structure of the Apertium translation pipeline, which consists of numerous small isolated applications (modules) that are connected using Unix pipes. The text from the source language (SL) is simply piped through these modules and the result will be the translation in the target language (TL). Apertium consists of the following modules[5]:

- *De-formatter*: Escapes formatting information in the text, e.g. HTML tags, so that only the text is considered.
- *Morphological analyzer*: Performs tokenization and morphological analysis which for a given surface form returns all of the possible lexical forms (analyzes) of the word.
- *Part-of-speech tagger*: POS tagger based on Hidden Markov models (HMM). Given a sequence of morphologically analysed words chooses the most likely sequence of POS tags.
- *Lexical selection*: A lexical selection based on constraint grammar [3] selects between possible translations of a word based on sentence context.
- *Lexical transfer*: For an unambiguous lexical form in the SL, this module returns the equivalent TL form based on a bilingual dictionary.
- *Structural transfer*: Performs local morphological and syntactic changes to convert the SL into the TL.
- *Morphological generator*: For a given TL lexical form, this module returns the TL surface form.
- *Post-generator*: Performs orthographic changes such as contractions and apostrophisation.
- *Re-formatter*: Restores escaped formatting information.

For each language pair, Apertium needs a monolingual SL dictionary used by the morphological analyzer, a bilingual SL-TL dictionary used by the lexical transfer module, a monolingual TL dictionary used by the morphological generator, and transfer rules used by the structural transfer module. Figure 1 depicts the structure of the Apertium platform.

[4]`http://wiki.apertium.org/wiki/List_of_language_pairs`
[5]List appears modified from [8]

3 IceNLP

IceNLP[6] is an open source NLP (Natural Language Processing) toolkit for analysing Icelandic text [7]. The toolkit is implemented in Java, and is licensed under the Lesser General Public Licence[7] (LGPL). Currently, IceNLP contains the following modules[8]:

- *Tokeniser*: Performs both word tokenization and sentence segmentation.
- *IceMorphy*: A morphological analyser [4]. The program provides the tag profile (the ambiguity class) for known words by looking up words in its dictionary. The dictionary is derived from the

  Icelandic Frequency Dictionary (IFD) corpus [12]. The tag profile for unknown words, i.e. words not known to the dictionary, is guessed by applying rules based on morphological suffixes and endings. IceMorphy does not generate word forms, it only carries out analysis.
- *IceTagger*: A linguistic rule-based POS tagger [4] which uses morphosyntactic tags from the tagset of the IFD corpus. The tagger uses IceMorphy for morphological analysis and applies both local rules and heuristics for disambiguation.
- *TriTagger*: A statistical POS tagger. This trigram tagger is a reimplementation of the well-known HMM tagger described by Brants [1]. It is trained on the IFD corpus.
- *Lemmald*: A lemmatiser [2]. The method used combines a data-driven method with linguistic knowledge to maximise accuracy.
- *IceParser*: A shallow parser. The parser marks both constituent structure and syntactic functions using a cascade of finite-state transducers [6].

Evaluation of IceTagger's tagging accuracy is 92.51% for the whole tag string[9] [5]. This is the state-of-the-art tagging accuracy for tagging Icelandic text.

Evaluation of IceParser's accuracy using F-measure[10] shows that the F-measure for constituents and syntactic functions is 96.7% and 84.3%, respectively, using 509 randomly selected sentences from IFD [6].

4 Using IceNLP with Apertium

As we stated in section 2, the Apertium translation pipeline consists of small independent modules that are linked using Unix pipes. This architecture allows one to easily modify the translation pipeline by replacing modules of interest.

In IS-EN the aim was to use IceTagger, Lemmald and IceParser in the Apertium pipeline (instead of the morphological module and the pos-tagger module).

[6]http://icenlp.sourceforge.net

[7]http://www.fsf.org/licensing/licenses/lgpl.html

[8]List appears modified from [8]

[9]This accuracy number is obtained by integrating TriTagger with IceTagger. Evaluations were based on the IFD corpus.

[10]The harmonic mean of precision and recall.

The reason was twofold. First, Apertium contains a HMM-based POS-tagger. Evaluations have shown that such taggers have lower accuracy when tagging Icelandic compared to IceTagger, the state-of-the-art POS-tagger for Icelandic [5]. Using a tagger with higher tagging accuracy should lead to better translation. Second, the emphasis was to use existing tools to a large extent.

When this research began, Apertium did not contain any module for syntactic analysis. It was a research question of interest whether adding IceParser to the translation pipeline would yield a better translation. Since then, a constraint grammar module, VISL CG3[11], developed outside the Apertium project, has been used in Apertium. This module can be used for syntactic analysis on the input to improve the translation results. Despite that, it is still an interesting research question whether VISL CG3 or IceParser yields a better translation results in IS-EN.

Initially, IceTagger and Lemmald were placed unmodified in the translation pipeline using a simple tag-mapper from the IFD tagset to the Apertium tagset. This was done by placing them at the front of the Apertium pipeline. As we stated in the introduction, this added considerable delay to the translation process due to the initial lexicon loading. The reason is that IceNLP was developed to handle large inputs and write out large outputs such as corpora. In these cases the time it takes to load the lexicon files into memory becomes relatively smaller than the time it takes to process the text resource and does not affect users. This is different when it comes to MT where the input text is usually short, then the initial loading time becomes larger than the translation time.

For using IceNLP efficiently with Apertium we had to eliminate this loading delay and add number of enhancements. For this we implemented a daemonized version of IceNLP that consists of client- and server applications that we call IceNLPServer and IceNLPClient. We will now go through these applications and discuss the modifications that we added to IceNLP.

4.1 Daemonizing IceNLP

The daemonized version of IceNLP consists of two applications: IceNLPServer and IceNLPClient. The idea is as follows. IceNLPServer contains an instance of the IceNLP toolkit, when started it stores all necessary lexicons files in memory and keeps them there during its execution. The server analyzes text, sent from clients without reloading the Lexicon files. We can then place IceNLPClient in the Apertium pipeline without the abovementioned loading delay.

Clients do not request usage of particular module in IceNLP. Instead users configure the output from the server using defined keywords. The server is aware of what module to use to fulfill the output requirements.

We will now discuss IceNLPServer and IceNLPClient in more depth both in terms of implementation and usage. We will then discuss our Mapping Lexicon and go through other modification that were added to IceNLP.

4.1.1 IceNLPServer

IceNLPServer is the server part in the daemonized version of IceNLP and has the role to process requests from IceNLPClient.

[11]`http://wiki.apertium.org/wiki/Apertium_and_Constraint_Grammar`

The processing phase of the server can be divided into the following three steps:

- Analyzing: When the server receives a request from a client, it start by analyzing the input using the tools from IceNLP.
- Mapping: When the Analyze phases finishes, the server goes through the results and applies mapping rules that users can define (see 4.2)
- Output generation: When the Mapping phase finishes the server creates the output reply that is sent back to the client.

IceNLPServer consists of the following modules:

- *Configuration*: Parses the IceNLPServer configuration file and provides a service layer to another modules for configuration informations.
- *Network thread*: The main thread in the server. This thread accepts new connections from IceNLPClients.
- *Client thread*: The network thread spawns a new client thread that handles the communication between the server and the client.
- *Output generator*: Formats the output from the server using the mapping lexicon described in 4.2

When the server is started it reads a configuration file. Through this configuration file, users can alter most of the server functionality, such as networking settings, output formation and IceNLP configuration. The following list contains some of the configuration option that IceNLPServer provides.

- *host*: Sets the host name of the server.
- *port*: Sets the port that the server listens on.
- *backlogSize*: Set the queue backlog size for the server socket.
- *IceTaggerLexiconsDir*: Set the location of the IceTagger lexicon files.
- *tokenizerlexicon*: Set the location of the tokenizer lexicon files.
- *OutputFormat*: Set the formation of the server response to clients. For Apertium this is set to: "ˆ[LEXEME][LEMMA][TAG]$". This means, for each lexeme in a sentence the analyzed output will begin on a ˆ sign, followed by the the lexeme, the lemma and the POS tag with a $ sign at the end. [LEXEME], [LEMMA], [TAG] are keywords and can be written in any order. Other keywords are [SUBJ] (Subject) and [OBJ] (Object) for syntactic purposes.
- *PunctuationSeparator*: Sets the separator between the tagging result and the previous tagging result when a word is annotated as linked word. By default the value for this is " ".
- *UserIceTaggerWhitespaceBlocks*: Uses the blocks that appear in the input text in the output text. That is, keeps the format of the input text.

- *MappingLexicon*: Location to a mapping lexicon file. The format of this file will be described in section 4.2.
- *TritaggerLexicon*: Set the location of the IceNLP TriTagger lexicons.
- *Tritagger*: Flag for using TriTagger with IceNLP.
- *debug*: Flag for setting the server to debug mode. Shows more output.
- *compiled_bidix*: Set the location of a compiled Bilingual Dictionary. Used for Apertium.

IceNLPServer was implemented under the is.iclt.icenlp.server package in the IceNLP source repository[12]. The repository contains an example configuration file for IceNLPServer[13]. We use that configuration file for our IS-EN MT system.

To start IceNLPServer, change directory to server/sh and execute the RunServer.sh script.

4.1.2 IceNLPClient

IceNLPClient is a small application that reads text from the standard input and sends it to IceNLPServer for processing. The results from IceNLPServer are then written to the standard output in the format that is configured in the server.

The following listing is an example how IceNLPClient is used.

```
hlynur@piro:~/icenlp/server/sh$ echo "Hundurinn geltir" |
 ./RunClient.sh
^Hundurinn/hundur<n><m><sg><nom><def>$ ^geltir/gelta<vblex>
<actv><pri><p3><sg>$
```

Listing 1: Using IceNLPClient

In the above listing the server has the Apertium output format defined in the configuration list in section 4.1.1. When the client is used, it expects the host/-port to connect to. This can be altered in the RunClient.sh shell script.

As we stated above, we use IceNLPClient in the Apertium pipeline. Figure 2 depicts the Apertium pipeline where IceNLPClient has been added to it.

IceNLPClient was implemented under the is.iclt.icenlp.client package in the IceNLP source repository[14].

4.2 Mapping Lexicon

A Mapping Lexicon was added to IceNLP to do various mappings on the output generated by IceTagger. This was needed to mimic the functionality of the Apertium morphological analyser/POS tagger. All mappings were implemented by a single mapping file with different sections for different purposes. We will now go through these mapping sections and describe their meaning. The Mapping lexicon the is used in IS-EN can be found in the IceNLP repository[15].

[12] `https://icenlp.svn.sourceforge.net/svnroot/icenlp/server`
[13] `https://icenlp.svn.sourceforge.net/svnroot/icenlp/server/configs/server.conf`
[14] `https://icenlp.svn.sourceforge.net/svnroot/icenlp/server`
[15] `https://icenlp.svn.sourceforge.net/svnroot/icenlp/core/dict/icetagger/otb.apertium.dict`

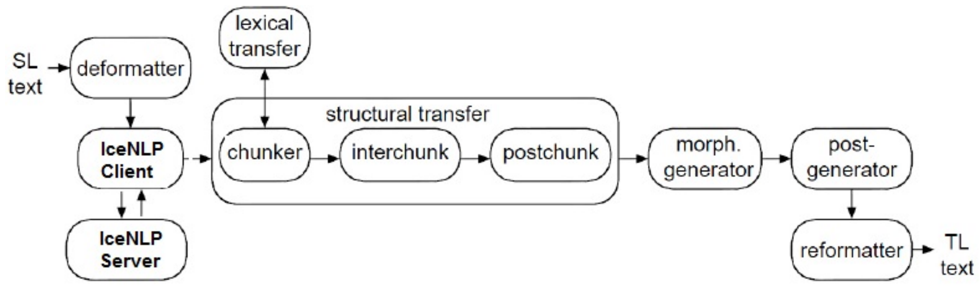


Figure 2: The modular architecture of the Apertium platform using IceNLP Client

- *TAGMAPPING* contains rules to map IFD tags to tags in another tagset. The following listing is an example where a TAGMAPPING rule are used.

  ```
  [TAGMAPPING]
  ...
  tfhee <num><nt><sg><gen>
  ```

  Here we are mapping the IFD tag "tfhee" to the Apertium tag "<num><nt><sg><gen>".

- *LEMMA* is used to map tags regarding exceptions for a particular lemmata. The following listing is an example where LEMMA rules are used.

  ```
  [LEMMA]
  ...
  vera <vblex><actv> <vbser>
  hafa <vblex><actv> <vbhaver>
  ```

  The above entries show that after tag mapping, the tags <vblex><actv> (verb, active voice) for the lemmata “vera” ’to be’ and “hafa” ’to have’ should be replaced by the single tag <vbser> and <vbhaver>, respectively. The reason is that Apertium needs specific tags for these verbs.

- *LEXEME* is used to map tags regarding exceptions for a particular lexeme. The following listing is an example where LEMMA rules are used.

  ```
  ...
  afar <adv> <preadv>
  mjög <adv> <preadv>
  ```

- *MWE* contains rules to map multiword expressions to a single tag. IceTagger tags each word of a MWE, whereas Apertium handles them as a single unit because MWEs cannot be translated word-by-word. The following listing is an example where MWE rules are used.

  ```
  [MWE]
  ...
  að_einhverju_leyti <adv>
  af_hverju <adv><itg>
  ```

4.3 White space memory

After the tokenizer had tokenized input text for IceNLP, the original format of the text was lost. If the text contained multiple new lines or additional white spaces between words then they were discarded. For keeping the original format of the text when IS-EN was used we added white space memory into the IceNLP tokenizer. Now, each lexeme remembers the spaces that came before them in the text.

5 Scalability of IS-EN

One of the requirement was to make IS-EN accessible on-line for debugging and demonstration, and as well to expose IS-EN as a web-service for external application usage. One can see that neither our client/server solution, nor standalone Apertium, does handle growing amounts of users in such context.

To meet this requirement, we implemented a scalable router/slave solution where the router receives translation requests and routes them to available translation slaves. Each slave contains an instance of the Apertium platform, the is-en language pair and IceNLPServer/IceNLPClient.

For communicating with the router we implemented a library which we call ARC (Apertium Router Client) that is used to send and receive translations from the router.

Note that there exist two similar scaling frameworks for Apertium, Apertium SOA [11] and ScaleMT [13]. We decided to build our own solution from scratch, because we were creating a "Frankenstein"[16] language pair for Apertium and such language pairs have never been used with either of the available solutions. It would be interesting to try our language pair with these framework, but it was out of the scope for this study.

We will now discuss our router/slave solution and how it is used.

5.1 Router

Router is a scalable solution, written in Java, to expose IS-EN on-line. The idea is as follows. We have a single router that listens to incoming connections on two ports, one for ARC clients that are sending translation requests and another for incoming slaves which are able to serve translation requests.

The role of the router is to route incoming translation requests to slaves and do so in some intelligent fashion to distribute the load between slaves. To start the router, change directory to sh and run the RunRouter.sh shell script. If the router starts normally then it will print the following on the screen:

```
>> IceNLP Router
[RouterRunner]: hostname set to 192.168.1.2
[SlaveListeningThread]: Listening on port 2525
[RequestListneningThread]: Listening on port 2526
[RequestListneningThread]: waiting for connections.
```

Listing 2: Starting the router

[16]System made of Apertium modules and modules from other systems[9].

This means that the router is listening for incoming slaves on port 2525 and for translation requests on port 2526. To change the ports that the server is listening on or setting the host name for the server, edit the RunRouter.sh shell script or run the router with -help flag.

The router can be configured to handle translation request itself if there are no available slaves for translation.

5.2 Slave

The slave is a client that connects to the router and waits for translation commands. When a translation arrives then the slave executes IS-EN and returns the reply back to the router which is then returned back to the client that sent the translation request. Note that it is not required that the slave is running on the same machine as the router. The load can therefore be distributed over number of machines.

To start a slave run the RunSlave.sh from the sh directory. Note that the Slave.jar needs to communicate with Apertium and the is-en language pair. Thus the machine that is running the slave must have an instance of Apertium. This can be configured in the slave configuration file under the configs folder in server[17]. If a slave starts normally then it will print the following on the screen:

```
[SlaveRunner]: connecting to 192.168.2.1:2525
[SlaveRunner]: ready.
```

Listing 3: Starting the router

This means that the *slave* is connected to *router* 192.168.2.1:2525 and is ready to serve translation requests.

5.3 ARC

ARC is a Java library code which was developed for allowing other applications to communicate with the router. Developers can include the server jar file in their project and use this library to connect to a router. Listing 4 is an example of such usage.

```
NetworkHandler handler = new NetworkHandler(hostname, port);
String res = apertiumHandler.translate(translationText);
```

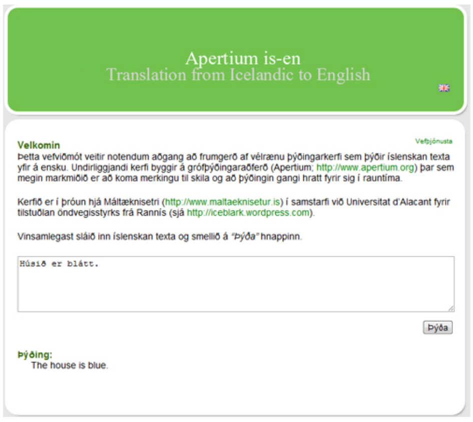
Listing 4: Using ARC in Java code

We also implemented a simple Java GUI which uses the ARC library. To run the GUI, change directory into the sh folder and run the RunARCGui.sh shell script.

6 Making IS-EN available online

For making IS-EN available on-line for debug and demonstration, we created a form based website where users can enter text for translation and a simple web service that users can use in their applications.

[17]https://icenlp.svn.sourceforge.net/svnroot/icenlp/server/configs/slave.conf


Apertium is-en
Translation from Icelandic to English

Velkomin

Vefþjónusta

Þetta vefviðmót veitir notendum aðgang að frumgerð af vélrænu þýðingarkerfi sem þýðir íslenskan texta yfir á ensku. Undirliggjandi kerfi byggir á grófþýðingaraðferð (Apertium; http://www.apertium.org) þar sem megin markmiðið er að koma merkingu til skila og að þýðingin gangi hratt fyrir sig í rauntíma.

Kerfið er í þróun hjá Máltæknisetri (http://www.maltaeknisetur.is) í samstarfi við Universitat d'Alacant fyrir tilstuðlan öndvegisstyrks frá Rannís (sjá http://iceblark.wordpress.com).

Vinsamlegast sláið inn íslenskan texta og smellið á *"Þýða"* hnappinn.

Húsið er blátt.

Þýða

Þýðing:
The house is blue.

Figure 3: Screenshot of IS-EN web form.

Figure 3 shows screenshot of the website. It has been deployed on the the nlp.cs.ru.is website[18]

We also implemented a web service that accepts HTML POST requests which contain the Icelandic text string that one wishes to translate. This web service returns a JSON object that contains the English translation. This web service has been deployed on nlp.cs.ru.is as well.

This allows users using other languages then Java to communicate with IS-EN over the internet. Listing 5 is an example of how this webservice can be used with the Python[19] programming language.

```
#!/usr/bin/python
# coding=UTF-8
import urllib, urllib2

# Text to translate.
text = "Hundurinn geltir"

# Location of the webservice.
url = 'http://nlp.cs.ru.is/ApertiumService/translate'

# Encode the parameters.
parameters = {'text' : text}
data = urllib.urlencode(parameters)
request = urllib2.Request(url, data)

# This request is sent in HTTP POST.
response = urllib2.urlopen(request)
data = response.read()

# Print out the translation.
print data
```

Listing 5: Using IS-EN webservice with Python

[18]http://nlp.cs.ru.is/is-en.htm
[19]http://www.python.org/

Both the website and the webservice are using the ARC client for communicating the router/slave.

7 Acknowledgements

Hrafn Loftsson and Martha Dís Brandt for a good collaboration on the IS-EN project and #apertium community on the freenode irc server for late night chats and informations on the Apertium platform.

8 Conclusion

We have described a Daemonized version of the IceNLP toolkit and modification that were added to it for the purpose of using it with the Apertium platform for doing machine translation from Icelandic to English.

References

[1] Thorsten Brants. TnT: A statistical part-of-speech tagger. In *Proceedings of the* 6th *Conference on Applied Natural Language Processing*, Seattle, WA, USA, 2000.

[2] A. K. Ingason, S. Helgadóttir, H. Loftsson, and E. Rögnvaldsson. A Mixed Method Lemmatization Algorithm Using Hierachy of Linguistic Identities (HOLI). In B. Nordström and A. Rante, editors, *Advances in Natural Language Processing,* 6th *International Conference on NLP, GoTAL 2008, Proceedings*, Gothenburg, Sweden, 2008.

[3] Fred Karlsson, Atro Voutilainen, Juha Heikkilä, and Arto Anttila. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text.* Mouton de Gruyter, Berlin, 1995.

[4] Hrafn Loftsson. Tagging Icelandic text: A linguistic rule-based approach. *Nordic Journal of Linguistics*, 31(1):47–72, 2008.

[5] Hrafn Loftsson, Ida Kramarczyk, Sigrún Helgadóttir, and Eiríkur Rögnvaldsson. Improving the PoS tagging accuracy of Icelandic text. In *Proceedings of the* 17th *Nordic Conference of Computational Linguistics (NODALIDA-2009)*, Odense, Denmark, 2009.

[6] Hrafn Loftsson and Eiríkur Rögnvaldsson. IceParser: An Incremental Finite-State Parser for Icelandic. In *Proceedings of the* 16th *Nordic Conference of Computational Linguistics (NoDaLiDa 2007)*, Tartu, Estonia, 2007a.

[7] Hrafn Loftsson and Eiríkur Rögnvaldsson. IceNLP: A Natural Language Processing Toolkit for Icelandic. In *Proceedings of Interspeech 2007, Special Session: "Speech and language technology for less-resourced languages"*, Antwerp, Belgium, 2007b.

[8] Hrafn Loftsson, Hlynur Sigurþórsson, and Francis M. Tyers Martha Dís Brandt. Apertium-IceNLP: A rule-based Icelandic to English machine translation system. Submitted, 2010.

[9] Gema Ramírez-Sánchez Mikael L. Forcada, Francis M. Tyers. The apertium machine translation platform: Five years on. In *Proceedings of the First International Workshop on Free/Open-source Rule-Based Machine Translation*, pages 3–10, 2009.

[10] Sergio Ortiz Rojas Juan Antonio Pérez Ortiz Mikel L. Forcada, Boyan Ivonov Bonev. Documentation of the Open-Source Shallow-Transfer Machine Translation Platform Apertium. Technical report, Department of de Llenguatges i Sistemes Informátics, Universitat d' Alacant, 2010.

[11] Pasquale Minervini. Apertium goes SOA: An efficient and scalable service based on the Apertium rule-based machine translation platform. In *Proceedings of the First International Workshop on Free/Open-Source Rule-Based Machine Translation*, Alacant, Spain, 2009.

[12] Jörgen Pind, Friðrik Magnússon, and Stefán Briem. *Íslensk orðtíðnibók [The Icelandic Frequency Dictionary]*. The Institute of Lexicography, University of Iceland, Reykjavik, 1991.

[13] Víctor M. Sánchez-Cartagena and Juan Antionio Pérez-Ortiz. ScaleMT: a Free/Open-Source Framework for Building Scalable Machine Translation Web Services. *Prague Bulletin of Mathematical Linguistics*, 93:97–106, 2010.