An efficient and scalable service-oriented architecture for the Apertium rule-based machine translation platform

Pasquale Minervini
Dipartimento di Informatica
Università degli Studi di Bari
Via E. Orabona 4, 70125 Bari, Italy
p.minervini@gmail.com

Abstract

Service Oriented Architecture (SOA) is a paradigm for organising and using distributed services that may be under the control of different ownership domains and implemented using various technology stacks. In some contexts, an organisation using an IT infrastructure implementing the SOA paradigm can take a great benefit from the integration, in its business processes, of efficient machine translation (MT) services to overcome language barriers. This paper describes the architecture and the design patterns used to develop an MT service that is efficient, scalable and easy to integrate in new and existing business processes. The service is based on Apertium, a free/open-source rule-based machine translation platform.

1 Introduction

Service Oriented Architecture is an architectural paradigm providing a set of principles of governing concepts used during phases of systems development and integration. In such an architecture, functionalities are packaged as interoperable, loosely coupled services that may be used to build infrastructures enabling those with needs (consumers) and those with capabilities (providers) to interact across different domains of technology and ownership.

Several new trends in the computer industry rely upon SOA as their enabling foundation, including the automation of Business Process Management (BPM) and the multitude of new architecture and design patterns generally referred to as Web 2.0 (O'Reilly, 2005).

In some contexts, an organisation using an IT infrastructure implementing the SOA paradigm can take a great benefit from the integration, in its business processes, of an efficient machine translation service to overcome language barriers; for instance, it could be integrated in collaborative enviroments where people, who have no language in common, attempt to communicate with each other; or in knowledge extraction processes, where data is not available in a language that can be understood by the domain experts or the knowledge extraction tools being used.

We implemented a machine translation and language recognition service by relying on Apertium[1] (Armentano-Oller et al., 2005), a free/open-source rule-based machine translation platform, and on libTextCat[2], a library implementing n-gram based text categorisation (Cavnar and Trenkle, 1994), which provides an inexpensive and highly effective way of recognising the language used in documents. libTextCat uses small-sized fingerprints of the desired languages (circa $4KB$ each) rather than resorting to more complicated and costly methods such as natural language parsing or assembling detailed lexicons; it is also used by Bitextor (Esplà-Gomis, 2009), a system to har-

[1] http://www.apertium.org/

[2] http://software.wise-guys.nl/libtextcat/

vest translation memories from multilingual websites.

Our decision to prefer a rule-based machine translation system like Apertium to a statistical or an example-based machine translation system was motivated by the following reasons:

- Statistical Machine Translation systems tend to produce text that appears more "natural" than that produced by Rule-Based ones, with the result that "fluency" can outweight "fidelity", but a natural and fluent translation is not necessairly completely faithful to the original text;

- In Rule-Based Machine Translation systems, linguistic knowledge can be encoded explicitly in the form of linguistic data, so that both humans and automatic systems can process it (this feature can be a great benefit when using domain-specific linguistic knowledge);

- Rule-Based Machine Translation systems tend to produce more "mechanical" translations, so their errors tend also to be more evident;

- Experts who have designed a Rule-Based Machine Translation system find it much easier to diagnose and repair sources of translation errors, like wrong rules in modules or wrong entries in dictionaries.

Efficiency and scalability are critical for the service since, especially in collaborative enviroments, it should be able to sustain a heavy load of traffic. In this paper, the techniques and design patterns used to implement the machine translation service will be described and it will be compared to other existing machine translation systems.

2 Service APIs

Our service provides the two following capabilities:

Translation – for automatic translation of free text from a source language to a destination language;

Language recognition – to automatically guess the language used in a text;

In SOA, interoperability between services is achieved by using standard languages for the description of service interfaces and the communications among services. A widely accepted technique for implementing SOA consists in making use of Web Services (Erl, 2005); a Web Service is defined by the W3C as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." (Brown and Haas, 2004).

Alternative standards to SOAP are XML-RPC (Winer, 1999), a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism, and Representational State Transfer (REST) (Fielding, 2000), a style of software architecture for distributed hypermedia systems such as the World Wide Web.

parameters	`text`
	`source language`
	`destination language`
returns	`translation`
	`detected source language`

Table 1: Parameters and return value(s) for the **Translate** method.

parameters	`text`
returns	`detected language`

Table 2: Parameters and return value(s) for the **Detect** method.

Our service natively provides a XML-RPC interface to the translation and language recognition functionalities, and we also implemented SOAP and REST wrappers to it. All the interfaces follow the schema outlined in tables 1 and 2 to expose, respectively, the translation and the language detection functionalities; those can be subsumed by the following methods:

Translate – which receives three parameters called `text`, `source language` and

```
>>> import xmlrpclib
>>> proxy = xmlrpclib.ServerProxy
        ('http://xixona.dlsi.ua.es:8080/RPC2')
>>> print proxy.translate("Test for the machine
        translation service", "en", "es")
        ["translation"]
Prueba para el servicio de traducción automática
```

Figure 1: Example – invoking our service from the Python shell using XML-RPC.

`destination language` containing, respectively the text to be translated, the source language and the destination language, and returns a `translation` value containing the translated text; if the source language is omitted, then language recognition is used to guess it, and the guessed language is returned in the `detected source language` value.

Detect – which receives three parameters called `text` containing free text, and returns a `detected language` value containing the language used by the text.

In addition, our service provides a **Language Pairs** method that returns a sequence of all the language pairs supported by the translation system, each represented by a pair containing the corresponding `source language` and the `destination language`.

In all methods, languages are represented by their ISO 639-1 (ISO:639-1, 2002) code. Figure 1 shows a short example of how our service's XML-RPC interface can be invoked from the Python[3] shell.

3 Internal architecture of the service

Apertium is a transfer-based machine translation system which uses finite-state transducers for lexical processing, hidden Markov models (HMMs) for part-of-speech tagging and finite-state-based chunking for structural transfer. Its translation engine consists of an *assembly line*, composed by the following modules:

Formatters – which handle format-specific information with respect to text to be translated;

Morphological analyser – which tokenizes the text in *surface forms* and delivers, for each surface form, one or more *lexical forms* consisting of lemma, lexical category and informations about morphological inflection;

Part-of-speech tagger – which chooses one of the analyses of an ambiguous word, according to its context;

Lexical transfer module – which reads each lexical form of the surface form and delivers the corresponding destination language lexical form;

Structural transfer module – which detects and processes patterns of words that need special processing due to grammatical divergences between two languages;

Morphological generator – that, from a lexical form in the destination language, generates a suitably inflected surface form;

Post-generator – that performs some orthographic operations in the destination language such as contractions;

Actually, in Apertium those functionalities are wrapped in two libraries, called `liblttoolbox` and `libapertium`, and they are used to implement a set of *console programs* managing their input and output in the form of *text streams*. The console programs are then assembled by using a *UNIX pipeline* to implement a final console program, called `apertium`, which, given a language pair, handles a translation process in its entirety. All the informations required to execute a translation task associated to a language pair are contained in a *mode file*, which specifies which modules should be run, their parameters and order.

Our service has been realized in the form of a multithreaded program, which relies on `liblttoolbox` and `libapertium` to execute each step of a translation process.

To prevent the frequent acquisition and release of the resources required to execute each step of the aforementioned assembly line, our service makes use of the *pooling pattern* (Kircher and Jain, 2004), in which multiple instances of one type of resource are managed in a pool. This

[3] `http://www.python.org/`

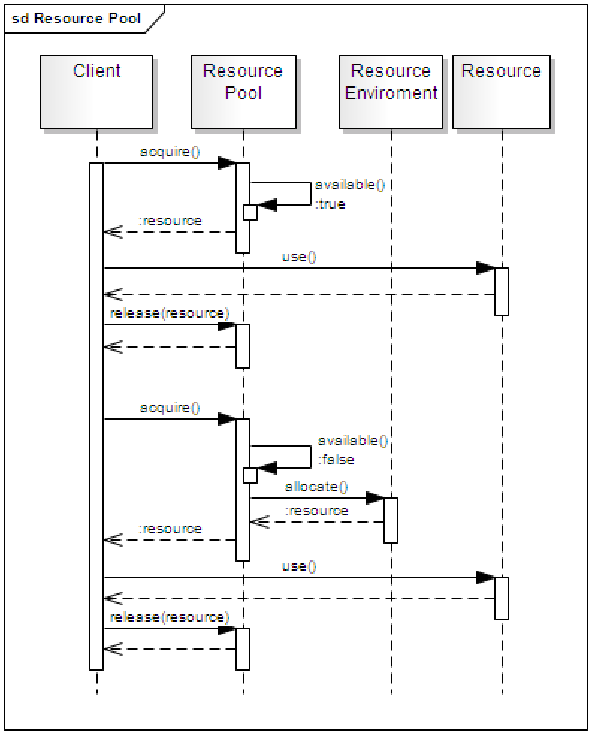

Figure 2: Sequence diagram describing how acquisition and release of resources works in a system implementing the *pooling pattern*: recycled objects are managed in a pool of resources, which allows pool clients to acquire them, and release them back to the pool when they are no longer needed.

pool of resources allows for reuse when resource clients release resources they no longer need: released resources are put back into the pool and made available to resource clients needing them, as shown in figure 2.

To improve efficiency, the resource pool can eagerly acquire a number of resources after its creation; then, if demand exceeds the number of available resources in pool, more resources can be *lazily* acquired.

There are various valid approaches to free unused resources, like those consisting of monitoring the use of a resource and controlling its lifecycle by using strategies such as "least recently used" (LRU) or "least frequently used" (LFU), or introducing a *lease* for every resource that specifies a time duration for which a resource can remain in the pool.

In our service, the default policy is to allocate new resources from the resource enviroment if there are no resources of the requested type available in the pool; the service also allows the setting of a *high water mark*, i.e. a maximum number of allocated objects: if the number of allocated objects is equal to the high water mark, the requesting client has to wait in a queue until a resource of the requested type is available in the pool. In addition, as we made no prior assumptions about how the service would be used, it does not apply any garbage collection policy by default.

Relying on a resource pool is designed to result in the following improvements for our rule-based machine translation service:

Performance – By preventing repetitious acquisition, elaboration and release of resources;

Predictability – Because direct acquisition of a resource from an external resource enviroment (for example, a filesystem or a DBMS) can lead, in some cases, to unpredictable results and dynamic memory allocation and deallocation can be non-deterministic with respect to time (Douglass, 2002);

Stability – Because repetitious acquisition and release of resources can increase the risk of system instability due, for example, to memory fragmentation problems (Utas, 2005; Douglass, 2002);

Scalability – As resources can be used also in different types of translation tasks, avoiding the allocation of a complete set of resources for each different translation task (for example, translation tasks on different language pairs using different dictionaries can make use of the same resource for managing the removal and restoration of formatting).

Another approach to implement a service based on Apertium by Sánchez-Cartagena and Pérez-Ortiz (2009) consists in making use of a pool of `apertium` processes: each translation request is routed to a process making use of the required language pair, and then its output is returned back to the service client.

Our approach has a series of pros and cons with respect to the one followed by Sánchez-Cartagena and Pérez-Ortiz (2009); advantages can be summarized by the following:

Efficiency – Threads usually require less resources when compared to processes, and

Inter-Process Communication (IPC) between multiple processes tend to be more complex and expensive than IPC between multiple threads belonging to the same process (Tanenbaum, 2007);

Scalability – Resources can be shared between multiple translation tasks (even belonging to different language pairs) without the need of allocating them for each translation process;

Portability – Our service relies on the Boost C++ libraries[4] for portable multithreading, regular expressions, filesystem operations and so on, making it capable to run in enviroments still not supported by the `apertium` application;

While disadvantages can be synthesized by the following:

Maintainability – Apertium internals still lack standardized API interfaces, therefore future changes to `liblttoolbox` and `libapertium` can make updates to our service necessary;

4 Results

To evaluate the efficiency of our service, which we will refer to as `apertium-service`, we compared the time it requires to compute and answer to a translation request from Spanish to English with the time required by the following systems:

- `apertium`, a console application implemented as a part of the Apertium project;
- `apertium-ws`, a REST service based on Apertium and described in Sánchez-Cartagena and Pérez-Ortiz (2009), using one *slave* instance attached to one *request router*;
- `moses`, a console application implemented as a part of Moses (Koehn et al., 2007), a Open Source Statistical Machine Translation system;
- `moses-service`, a service relying on Moses.

[4] `http://www.boost.org`

All the Apertium-based systems (`apertium`, `apertium-service` and `apertium-ws`) were employing the `apertium-en-es` language pair[5]. The translation model used by `moses` and `moses-service` has been trained on the well-known Europarl (Koehn, 2002) corpus by using SRILM (Stolcke, 2002), a toolkit for building and applying statistical language models. Language models have been then compiled into binary format using IRSTLM (Federico et al., 2008), and trained to minimize the error rate on a set of sentences from the same corpus.

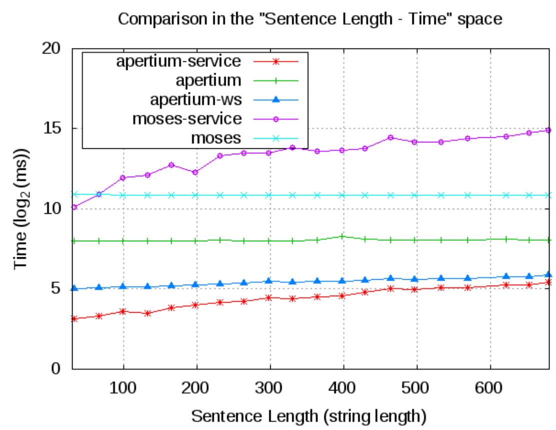

Figure 3: Comparison in the "Sentence Length - Time" space between `apertium-service`, `apertium`, `apertium-ws`, `moses` and `moses-service`; measurements are in $log_2(string\ length)$ for the Sentence Length dimension and in $log_2(ms)$ for the Time dimension.

All the experiments were run on a server with four 2GHz Dual-Core AMD Opteron processors and 4GB of main memory, using the GNU/Linux operating system. `apertium-service` and `moses-service` were accepting translation requests in the form of XML-RPC calls, `apertium-ws` in the form of REST HTTP GET requests, `apertium` and `moses` through standard input (a new process was created for each translation task). The free text used for timing all the systems was also taken from Europarl corpus. Figure 3 shows the time required to translate increasingly longer sentences for all systems (values in the time dimension are shown on a logarithmic scale), and figure 4 only for `apertium-service`, `apertium` and `apertium-ws`.

[5] SVN Revision 16218

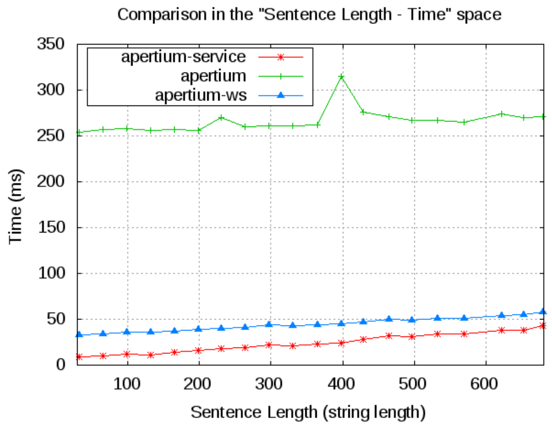

Figure 4: Comparison in the "Sentence Length - Time" space between `apertium` and `apertium-service`; measurements are in *string length* for the Sentence Length dimension and in ms for the Time dimension.

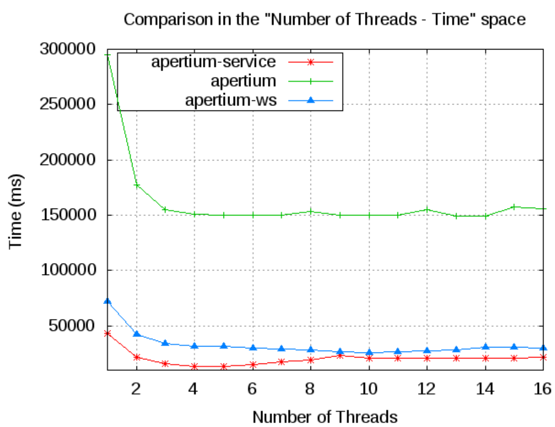

Figure 5: Comparison in the "Number of Threads - Time" space between `apertium` and `apertium-service`.

Scalability for `apertium` and `apertium-service` have been evaluated by calculating the average time required by the two systems to answer to 1,024 translations requests sequentially sent by a variable number of clients; the requests consisted to translating the longest sentence from the Europarl evaluation corpus (679 characters), so to obtain a worst case score, from Spanish to English. Figure 5 shows the results of this comparison.

5 Future work

Our service lends to a series of potentially useful applications:

UMLS concept identification in non-English medical documents: MetaMap (Aronson, 2001) is an application that allows mapping text to UMLS Metathesaurus[6] concepts, which have proved to be useful for many applications, including decision support systems, management of patient records, information retrieval and data mining within the biomedical domain.

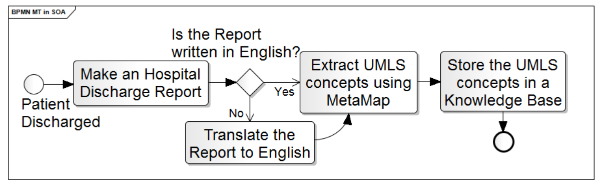

Figure 6: Representation of a business process in which a clinical document, if written in a language different than English, is fist translated to English and then processed using MetaMap to extract UMLS concepts.

Actually, MetaMap is only available for English free text, which makes it difficult the use of UMLS Metathesaurus to represent concepts from biomedical documents written in languages other than English. To enable cross-lingual text classification, Carrero et al. (2008) proposes to make use of general pourpose statistical machine translation tools, such as Google Translate[7], to translate the documents from their source language to English, and then process them as illustrated in Figure 6; unluckily, this approach presents some important mistakes when translating terms specific for the biomedical domain.

To overcome this limitation, it should be possible to employ our Apertium-based service, in conjunction with bilingual dictionaries, transfer rules etc. specific for the biomedical domain, to obtain an accurate translation of biomedical documents and then process them with a traditional English version of MetaMap.

Supporting creation of user-generated content: Wikipedia is an online, multilingual, volunteer-edited encyclopedia. "There are currently 262 language editions of Wikipedia; of these, 24 have over 100,000 articles and 81 have

[6]The UMLS Metathesaurus (Schuyler et al., 1993) provides a representation of biomedical knowledge consisting of concepts classified by semantic type and both hierarchical and non-hierarchical relationships among the concepts.

[7]`http://translate.google.com/`

over 1,000 articles" (Wikipedia, 2009). Although access to technology is also an important factor, the number of available articles in a particular language's Wikipedia corresponds somewhat to the number of available speakers of that language.

In many cases, closely related languages are mutually intelligible (Tyers et al., 2009), and even a prototype Machine Translation system can produce accurate translations (Armentano-Oller and Forcada, 2006). This seems to be the case with Nynorsk and Bokmål (Unhammer et al., 2009), where users of the Nynorsk Wikipedia have made contributions to the system's lexicon, to assist in their translation of articles from the larger Bokmål Wikipedia to the Nynorsk Wikipedia.

However, the current use of Machine Translation on the Wikipedias of marginalized languages is a somewhat error-prone process, where the original text is manually copied from the source Wikipedia, translated off-line, and pasted as a new article to the target Wikipedia. By providing an efficient, easily integrated service, we hope to remove some of the accidental errors inherent to this process.

In addition, the service's logging facilities may be used to improve Machine Translation quality, by incorporating user feedback (Chin and Rosart, 2008), in the form of corrections to the translated text. Small corrections to Wikipedia articles have been used in the construction of error corpora (Miłkowski, 2008), which can then be used to augment translation rules, or in the creation of statistical post-correction systems (Dugast et al., 2007).

6 Conclusions

We presented `apertium-service`, a machine translation service based on Apertium, a free/open-source rule-based machine translation platform. It has been shown to be competitive compared to other systems in both efficiency and scalability.

Source code for our service is released under the GNU General Public Licence version 3[8] and is available on the Apertium SVN repository.[9]

[8] `http://www.gnu.org/licenses/gpl.html`

[9] `http://apertium.svn.sourceforge.net/svnroot/apertium/trunk/apertium-service`

Acknowledgements

Development for this project was funded as part of the Google Summer of Code[10] programme. Many thanks go to Jimmy O'Regan, Francis Tyers and others involved in the Apertium Project, for their constant help. Additionally I am grateful to the anonymous reviewers for their invaluable comments and suggestions on an earlier version of this paper.

References

Armentano-Oller, C., Corbí-Bellot, A. M., Forcada, M. L., Ginestí-Rosell, M., Bonev, B., Ortiz-Rojas, S., Pérez-Ortiz, J. A., Ramírez-Sánchez, G., and Sánchez-Martínez, F. (2005). An open-source shallow-transfer machine translation toolbox: consequences of its release and availability. In *OSMaTran: Open-Source Machine Translation, A workshop at Machine Translation Summit X*, pages 23–30.

Armentano-Oller, C. and Forcada, M. L. (2006). Open-source machine translation between small languages: Catalan and aranese occitan. In *Strategies for developing machine translation for minority languages (5th SALTMIL workshop on Minority Languages)*.

Aronson, A. R. (2001). Effective mapping of biomedical text to the umls metathesaurus: the metamap program. *Proc AMIA Symp*, pages 17–21.

Brown, A. and Haas, H. (2004). Web services glossary. World Wide Web Consortium, Note NOTE-ws-gloss-20040211.

Carrero, F. M., Cortizo, J. C., Gómez, J. M., and de Buenaga, M. (2008). In the development of a spanish metamap. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 1465–1466, New York, NY, USA. ACM.

Cavnar, W. B. and Trenkle, J. M. (1994). N-gram-based text categorization. In *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175.

[10] `http://code.google.com/soc/`

Chin, J. and Rosart, D. (2008). Machine translation feedback.

Douglass, B. P. (2002). *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Dugast, L., Senellart, J., Simard, M., and Koehn, P. (2007). Statistical post-edition on systran rule-based translation system. In *ACL Workshop on Statistical Machine Translation*.

Erl, T. (2005). *Service-Oriented Architecture : Concepts, Technology, and Design*. Prentice Hall PTR.

Esplà-Gomis, M. (2009). Bitextor: a Free/Open-source Software to Harvest Translation Memories from Multilingual Websites. In *Proceedings of MT Summit XII*, Ottawa, Canada. Association for Machine Translation in the Americas.

Federico, M., Bertoldi, N., and Cettolo, M. (2008). Irstlm: an open source toolkit for handling large scale language models. In *Interspeech 2008*, pages 1618–1621. ISCA.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine.

ISO:639-1 (2002). Iso 639-1:2002 – codes for the representation of names of languages – part 1: Alpha-2 code.

Kircher, M. and Jain, P. (2004). *Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management*. Wiley.

Koehn, P. (2002). Europarl: A multilingual corpus for evaluation of machine translation. Draft.

Koehn, P., Hoang, H., Mayne, A. B., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. pages 177–180.

Miłkowski, M. (2008). Automated building of error corpora of polish. In Lewandowska-Tomaszczyk, B., editor, *Corpus Linguistics, Computer Tools, and Applications — State of the Art. PALC 2007*, pages 631–639. Internationaler Verlag der Wissenschaften.

O'Reilly, T. (2005). What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software.

Sánchez-Cartagena, V. M. and Pérez-Ortiz, J. A. (2009). An open-source highly scalable web service architecture for the apertium machine translation engine. In *Proceedings of the First International Workshop on Free/Open-Source Rule-Based Machine Translation*.

Schuyler, P. L., Hole, W. T., Tuttle, M. S., and Sherertz, D. D. (1993). The umls metathesaurus: representing different views of biomedical concepts. *Bull Med Libr Assoc*, 81(2):217–222.

Stolcke, A. (2002). Srilm – an extensible language modeling toolkit.

Tanenbaum, A. S. (2007). *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA.

Tyers, F. M., Wiechetek, L., and Trosterud, T. (2009). Developing prototypes for machine translation between two sámi languages. In *Proceedings of the 13th Annual Conference of the European Association of Machine Translation, EAMT09*.

Unhammer, K., Trosterud, T., and Tyers, F. (2009). Reuse of free resources in machine translation between nynorsk and bokmål.

Utas, G. (2005). *Robust Communications Software: Extreme Availability, Reliability and Scalability for Carrier-Grade Systems*. John Wiley & Sons.

Wikipedia (2009). Wikipedia — wikipedia, the free encyclopedia. [Online; accessed 28-September-2009].

Winer, D. (1999). XML/RPC specification. Technical report, Userland Software.