

Apertium goes SOA

An efficient and scalable service based on the Apertium rule-based machine translation platform

Pasquale Minervini Jimmy O'Regan

First International Workshop on Free/Open-Source Rule-Based
Machine Translation

Table of Contents

- 1 Introduction
 - The Joy of SOA
 - Why Rule-Based Machine Translation?
- 2 Service APIs
 - Technologies we rely on
- 3 Service Architecture
 - Resource pooling
 - Problems encountered
- 4 Results
- 5 Future Works

Service Oriented Architecture

- Service Oriented Architecture (SOA) is an architectural paradigm:
 - Functionalities are packaged as interoperable, loosely coupled services;
 - Service can be combined to define Business Processes;
- An organisation using SOA can take a great benefit from the integration of an efficient machine translation service to overcome language barriers;

Why Rule-Based Machine Translation?

- We decided to rely on a Rule-Based Machine Translation platform because:
 - Linguistic knowledge can be encoded explicitly (both humans and machines can process it);
 - Translation tend to be more faithful to the original meaning than in common SMT systems;
 - Errors tend to be more evident;

- Service's interface can be subsumed by the following two methods:
- **Translate:** `{ text, source language, destination language }`
→ `{ translation, detected source language }`;
 - If the `source language` parameter is omitted, language detection is used to guess it, and the `detected source language` return value is set to the guessed language.
- **Detect:** `{ text }` → `{ detected language }`;
- In all methods, languages are represented by their ISO 639-1¹ code.

¹International Organization for Standardization (2002) – ISO 639-1:2002, Codes for the representation of names of languages, Part 1: Alpha-2 code

- A widely accepted technique for implementing SOA consists in making use of Web Services²;
- A Web Service is defined by the W3C as “a software system designed to support interoperable machine-to-machine interaction over a network. [...] 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.”³;
- Alternative standards are:
 - XML-RPC – a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism
 - Representational State Transfer (REST) – a style of software architecture for distributed hypermedia systems.

²Erl, T. (2005) – Service-Oriented Architecture : Concepts, Technology, and Design

³W3C (2004) – Web Services Glossary

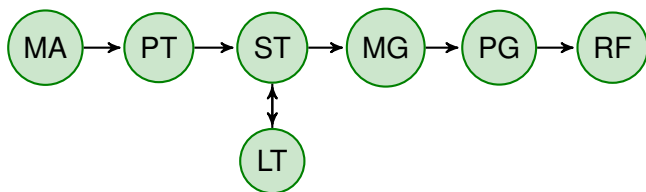
- **libTextCat**⁴ – a library implementing n-gram based text categorisation⁵, which provides an inexpensive and highly effective way of recognising the language used in documents; it uses small-sized fingerprints of the desired languages (circa 4KB each) rather than resorting to more complicated and costly methods such as natural language parsing or assembling detailed lexicons.
- **Apertium**⁶, a free/open-source rule-based machine translation platform.

⁴<http://software.wise-guys.nl/libtextcat/>

⁵William B. Cavnar and John M. Trenkle (1994) – N-Gram-Based Text Categorization

⁶<http://www.apertium.org/>

- Apertium is an *assembly line* of the following *modules*:



- Morphological Analyser – reads a FST generated from a SL morphological dictionary and delivers, for each *surface form*, one or more *lexical forms*;
- POS Tagger – reads a tagger definition and chooses the most likely lexical form corresponding to an ambiguous surface form;
- Lexical Transfer – reads a bilingual dictionary and turns each SL lexical form in a TL lexical form;
- Structural Transfer – reads a rule base and performs more operations on chunks; etc.

Resource Pooling

- Our service has been developed as a multithreaded C++ program which relies on functionalities implemented in the `libltttoolbox` and `libapertium` libraries to execute each step of the assembly line;
- Loading and elaborating all the resources needed to complete a translation task for each run is not an efficient solution;
- To prevent the acquisition and release of resources for each translation task, our service has been designed by making use of the *pooling pattern*:
 - It is desirable to keep all reusable, not currently in use resources in the same resource pool so that they can be managed by a coherent policy.

Resource Pooling (2)

- The resource pool enables resource reuse when resource clients release resources they no longer need: released resources are put back into the pool and made available to resource clients that needs them;
- It is a common practice to set a maximum number of resources that can be allocated by a system: we call this parameter `high water mark`;
- If the number of allocated resources is not less than the `high water mark` and the resource pool is empty, requesting clients are put in a *wait queue* until a resource of the requested type is available;

Resource Pooling (3)

- Relying on a resource pool provides a set of positive effects:
 - It prevents the repetitious acquisition, elaboration and release of resources → **performance**;
 - It minimizes memory fragmentation and the relying on an external resource environment → **predictability**⁷;
 - Resources can be shared among different kinds of translation tasks → **scalability**;
- It is possible to eagerly acquire a number of resources after the pool creation, and lazily acquire more if demand exceeds the number of available resources in pool;
- It is possible to employ many valid approaches to free unused resources (like LRU or LFU).

⁷Douglass, B. P. (2002) – Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems

Problems encountered

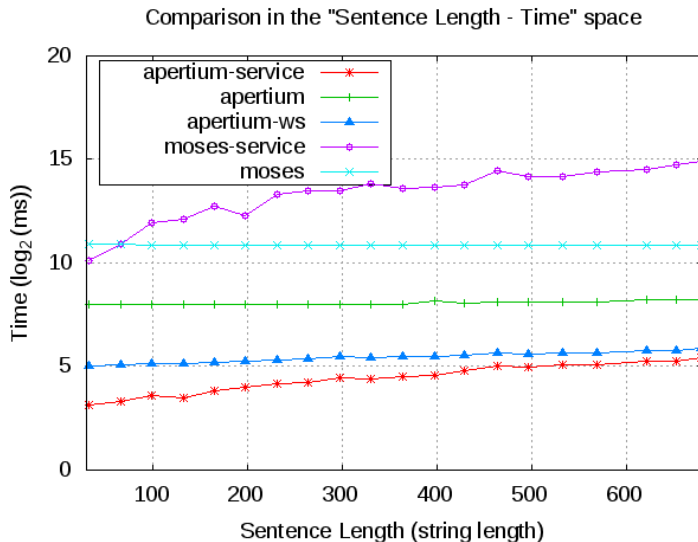
- Thread safety issues:
 - The `FSTProcessor` module had to be patched;
 - Formatters had to be rewritten (GNU Flex → Boost.Spirit);
- We had to deal specifically with every module's interface;
- Apertium modules uses on `C FILE` wide-oriented streams to receive the input text stream, but on some systems it is not possible to bind this kind of streams to memory buffers.

- Our service (`apertium-service`) has been compared with 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 a paper by Sanchez-Cartagena and Perez- Ortiz (2009)⁸, using one *slave* instance attached to one *request router*;
 - `moses`, a console application implemented as a part of Moses⁹, a Open Source Statistical Machine Translation system;
 - `moses-service`, a service relying on Moses.

⁸Sánchez-Cartagena, V. M. and Prez-Ortiz, J. A. (2009) - An open-source highly scalable web service architecture for the apertium machine translation engine.

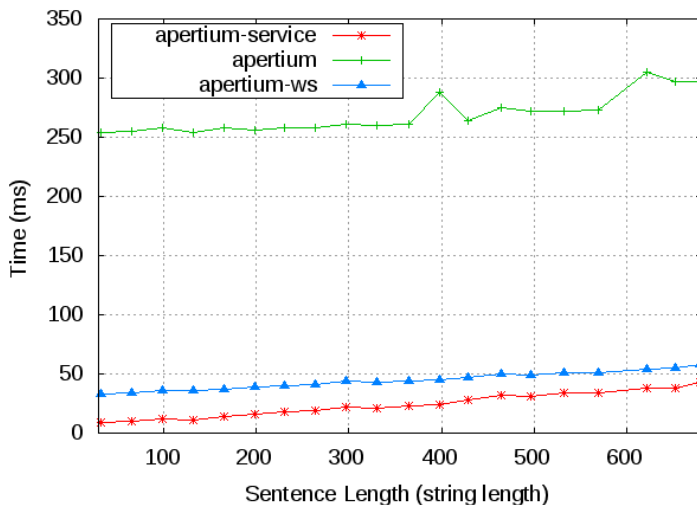
⁹Kohen, P. et al. (2007) - Moses: Open Source Toolkit for Statistical Machine Translation

Benchmarks



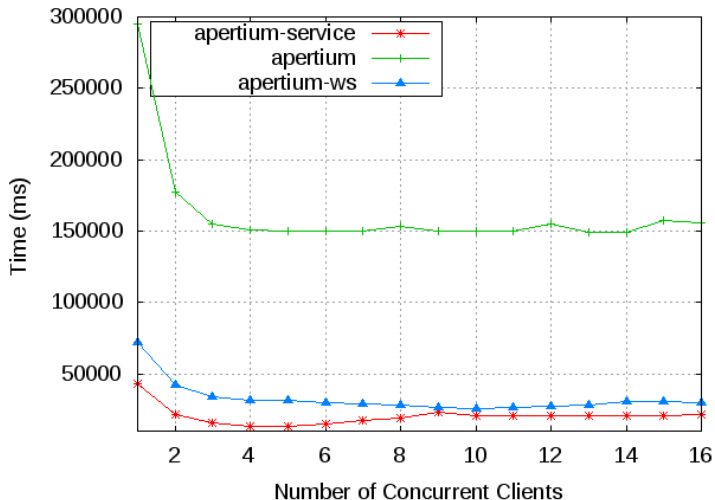
Benchmarks

Comparison in the "Sentence Length - Time" space



Benchmarks

Comparison in the "Number of Concurrent Clients - Time" space



Possible Future Developments

- More Free/Open-Source projects relying on Open Machine Translation services;
 - The Okapi¹⁰ frameworks already has a connector to `apertium-service`;
- A standardized interface hierarchy for Apertium modules, to make the development and integration of new modules easier;
- A common interface for Apertium-based services;

¹⁰<http://okapi.sourceforge.net/> – The Okapi Framework is a set of interface specifications, format definitions, components and applications that provides an environment to build interoperable tools for the different steps of the translation and localization process.