Distributed Translation Memories implementation using WebServices[0]

Alberto Simões
Linguateca
Universidade do Minho
ambs@di.uminho.pt

Xavier Gómez Guinovart
Semin. de Lingüística Informática
Universidade de Vigo
xgg@uvigo.es

José João Almeida
Projecto Natura
Universidade do Minho
jj@di.uminho.pt

Resumen: Las memorias de traducción son muy útiles para la traducción, pero son difíciles de compartir y de reutilizar por parte de la comunidad de traductores. Mediante las memorias distribuidas de traducción, los usuarios pueden construir conjuntamente y compartir memorias de traducción. En este artículo, presentamos los detalles de la implementación de esta arquitectura utilizando la tecnología de WebServices y un ejemplo de un sistema distribuido entre Portugal y España.
Palabras clave: memorias de traducción distribuidas, WebServices, TAO

Abstract: Translation Memories are very useful for translators but are difficult to share and reuse in a community of translators. This article presents the concept of Distributed Translation Memories, where all users can contribute and sharing translations. Implementation details using WebServices are shown, as well as an example of a distributed system between Portugal and Spain.
Keywords: distributed translation memories, WebServices, CAT

1 *Introduction*

Translation Memories (TM) are relations between sentences in two or more different languages. Translators use them to save old translations, hoping they can be reused later.

Common translation tools use TM, but use a proprietary format for their internal representation. Meanwhile, a standard for interchange of TM was defined. This standard is based on XML(XML, 10 February 1998) and is named *Translation Memory eXchange* (TMX) format (Savourel, 1997; OSCAR, 2003; Gómez, 2001).

Although useful, the use of TMX did not change the way translators work. They have their own TM and TMX is only used for interchange between big companies which have their own TM. Each translator of a community continues working only with their own computer: when translating a new sentence only local TMs are searched.

The idea behind Distributed Translation Memories (DTM) is to spread TMs on servers around the world where any translator could query for their sentences.

DTMs can be useful in two different situations:

- some big companies distribute Translation Memories to their translators, where technical phrases are already translated. In these cases, this company needs to send the TM by email, ftp or common mail to the translator. With DTM, these companies can build up a server to be used by their translators (identified by some username/password pair, for example);
- among a translators group working on the same project it is usual to have common sentences to translate. If they can share their own translation memories in a dynamic way, they can reuse them easily. With this in mind, each translation workstation will work as a local server like current peer-to-peer (P2P) share applications.

This article will explain how DTM can be used to help the translator, and how it can be implemented using simple technology.

2 *DTM impact on translators work*

From the translator's point of view, the utility of DTM available on line can enable them

[0]This work has been partially funded by the Ministerio de Ciencia y Tecnología (MCYT), the Fondo Europeo de Desenvolvemento Rexional (FEDER), the Xunta de Galicia, and the Universidade de Vigo, within the project "Linguistic-computational processing of the Linguistic Corpus of the University of Vigo (CLUVI)" (ref. BFF2002-01385), and Fundação para a Ciência e Tecnologia of Portugal through grant POSI/PLP/43931/2001, and co-financed by POSI.

to take advantage of the Internet resources in TM/CAT environments such as Trados or DéjàVu. Thanks to DTM, translators can manage very large volumes of TM databases without having them installed on their computers, saving their maintenance and update as well. On the one hand, translators can access DTM servers via Internet. These can be commercial servers (setup by translation companies) or free-access servers (such as the University of Minho's server or the University of Vigo's). On the other hand, translators can access DTM of their peer-to-peer community in order to share TM, for example, with the help of a Napster-style P2P program.

The search for a translation in this potentially enormous DTM on-line repository must increase substantially the productivity of TM/CAT programs because, as is obvious, the bigger the size of the enquired TM, the bigger the possibility of finding a translation where the translation unit in the source language is the same or similar as in the inquired text. As for free-access institutional servers of DTM, and regarding the public interest of their creation and effectiveness, we agree with Abaitua (Abaitua, 2001) in the need of creating a collective initiative in order to increase the size and consequently the utility of free-access DTM available on-line, with the help of persons and institutions related with the academic, publishing and professional translation fields.

To sum up, the features of the TM/CAT architecture oriented to on-line DTM can mean a very important improvement of the profitability of these systems. Firstly, the shared DTM can achieve an extension (and consequently a potential utility) bigger than the TM generally employed by users of these systems, as these TM are usually individual or transferred by a company for a specific translation. Secondly, the updating of these DTM available on line means an increase of the utility of the offered translations, a feature of these systems which can become particularly relevant in the event of P2P communities of translators assigned to the same project of translation. Finally, regarding the traditional utility of TM/CAT software, the distributed architecture means for translators a decrease of the needs of storage and computational processing. Also, along with the development of CAT environments on line, DTM technology allows to hand over the computational requirements to Internet, in terms of applications and resources, for the Translator's Workbench in a near future .

3 DTM Architecture

The architecture of Distributed Translations Memories depends to a great extent on their context of use.

In a translation community, translators use a workstation connected to a network where other translation workstations exist. This means that each workstation will act both as a client and as a server of DTMs. This is an architecture very similar to other peer-to-peer software. Figure 1 shows such an architecture.

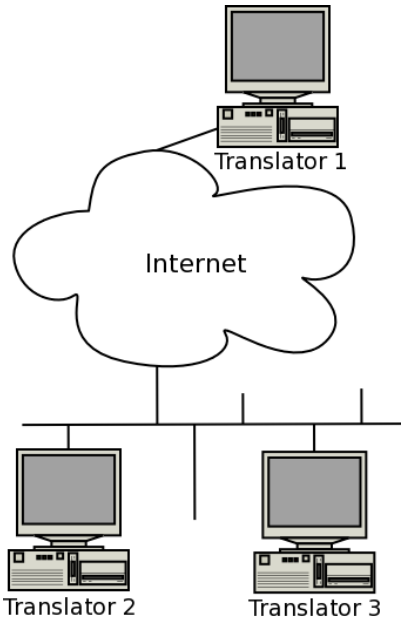

Figure 1: Peer-to-peer architecture

When the DTM is viewed as a service provided by a University, a translation software producer or other entities, the service provider will act only as a DTM server, while clients can be simple DTM clients or members in a peer-to-peer architecture. Figure 2 shows a client/server architecture.

The software implementation of these two services has different implications. The typical client-server architecture implementation relies in remote procedure call protocols[1] like RPC, Java RMI, Corba or, more recently, WebServices.

This type of implementation is not appropriate to the peer-to-peer architecture. As

[1]Or specific and closed implementations of communication.

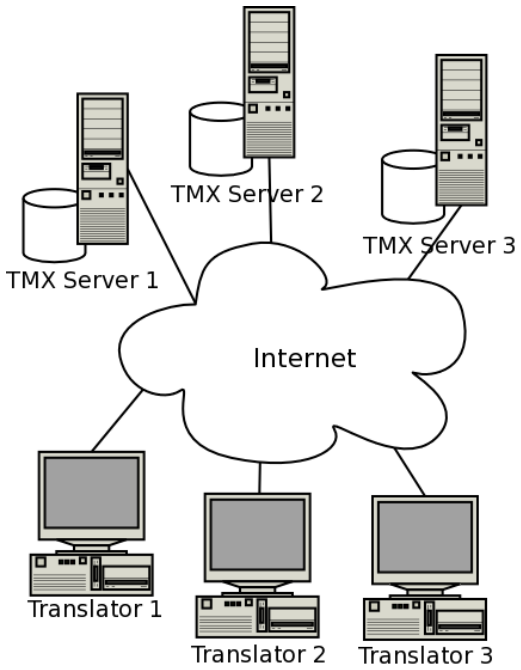

Figure 2: Client/server architecture

each machine will be client and server, and will run on a workstation, it is not common to have server daemons running on work machines.

Meanwhile, in this article we will rely on a web-service based implementation, for the client/server architecture.

4 DTM service definition

Web-services philosophy is based on a kind of Application Program Interface (API) to be used by the Internet. Each web-service server defines an Interface using WSDL (Web Service Description Language) (Christensen et al., 2001) used by the clients.

This interface is a set of functions definitions: arguments and their data types, and return data type. In this section we will define the functions or methods that a web-service server should implement.

4.1 Querying Languages

The ideal web-service server should be stateless: in most cases, servers cannot track users and store information about them. With this in mind, each message should be self-contained. Therefore, a message asking for a translation for a specific sentence should contain not only the sentence to be translated but also the language that sentence is in, and the language in which we want the translation.

This message could be sufficient but, in some cases, that would lead to clients asking for translations for languages the server does not include in its translation memories.

To solve this flood of messages, it is important to define a message to query the server for its knowledge on a pair of languages:

$$lang\_pair(\mathcal{L}_1, \mathcal{L}_2) \longrightarrow \mathcal{B}$$

The `lang_pair` message is sent with two language codes. The server will return a boolean value meaning its knowledge (or not) about that pair of languages.

Note that we are considering the translation memory to be reflexive: if the server can translate from language $\mathcal{L}_1$ to $\mathcal{L}_2$, then it can translate from $\mathcal{L}_2$ to $\mathcal{L}_1$.

4.2 Querying for a Translation

Although the normal translation software searches for a translation and uses only one of the found translations, in the implementation of DTM it is more important to give as many translations as possible at a time.

For that purpose, and keeping in mind the idea of a stateless server, we defined a message named `equiv` which returns equivalent sentences in another language:

$$equiv(\mathcal{L}_1, \mathcal{L}_2, \mathcal{S}_{\mathcal{L}_1}) \longrightarrow \mathcal{S}^{\star}_{\mathcal{L}_2}$$

The semantics of the server when answering this message is not so simple as it seems at a first glance: while sometimes there is the possibility of the query sentence to exist in the Translation Memory, the probability for that is very low. In fact, what translation systems based on TM do is to search for a similar sentence. The definition of the similarity function can be more or less complicated, depending on how we implement it. In the section 5.1.2 we discuss some possible implementations for this function.

4.3 Querying for Concordances

Another common functionality provided by CAT software is the concordance for a specific word or small sequence of words. It can be viewed as a specialized version of the `equiv` message. Differences rely on the fact that the server cannot align that word in the parallel text and as such it cannot answer with only one word either.

The API definition for this method will also have the two languages, and the words to

be queried as arguments. Meanwhile, the answer from the server will contain a sequence of sentence pairs: the sentence in the original language where the searched word occurs, and the aligned sentence in the target language.

$$conc(\mathcal{L}_1, \mathcal{L}_2, \mathcal{W}_{\mathcal{L}_1}) \longrightarrow (\mathcal{W}_{\mathcal{L}_1} \times \mathcal{W}_{\mathcal{L}_2})^\star$$

4.4 Contributing to TMs

TM servers can only survive if there is feedback from their users. The most useful feedback for TM servers are new TMs. Of course we can argue that to maintain TM quality on the servers, we cannot accept translations from just any translator. This is true, but following this idea, TM servers will never grow. We propose that contributed TMs should get into a stage area where they can later be examined for quality (using manual or automatic techniques).

Then, we defined a method for the TM contribution, and leave to the server admins to accept contributed translations or not.

$$submit(\mathcal{L}_1, \mathcal{L}_2, (\mathcal{S}_{\mathcal{L}_1} \times \mathcal{S}_{\mathcal{L}_2})^\star) \longrightarrow 1$$

5 Implementation Details

To test the idea and check for usability, we created two servers and two clients. One of each was implemented in Perl and `SOAP::Lite`, the other in PHP and `NuSOAP`. The Perl server and clients were developed in the Natura Project, at Braga, Portugal. The PHP server and clients were developed in Seminario de Lingüística Informática, at Vigo, Spain. This way, we could test two technologies and their interoperability.

5.1 Server Implementation

The basic idea behind the server implementation is to listen for messages and answer them, as expected. The main problem does not lie in the web-service implementation but in the way the translation memory is treated to give answers in acceptable time.

5.1.1 TM indexing

Indexing a Translation Memory is not simple. A format like TMX cannot be used, given the problems of indexing an XML file. Our approach is to use a textual indexer named Glimpse (Manber and Wu, 1994).

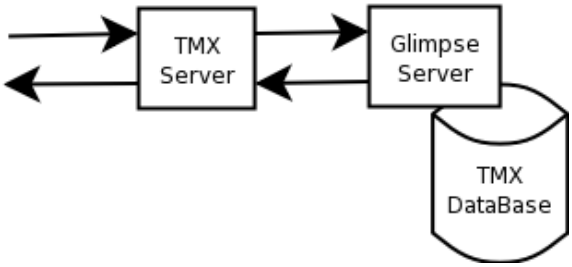

Figure 3: Glimpse-based server architecture

Glimpse indexes the text creating very quick indexes. They permit that a search for two words α and β will be performed only in the files the indexes say they exist. So, if we index many small files we have quicker indexes.

These indexes are big (normally two times the size of the indexed files), and as such takes a lot of time to load in memory. To solve this problem, the Glimpse distribution includes a server. It loads the indexes one time, and stays in memory accepting queries.

Our server implementation uses this feature. Each time the web-service receives a query (`conc` or `equiv`), it sends the query to the glimpse server, waiting for the answer and sending it again in the form of a web-service to the client. This architecture is shown on figure 3.

5.1.2 Similarity function implementation

This is one of the biggest problems of our implementation, at the moment. Using glimpse, we have two choices: to get the exact match (which does not offer many chances to find the sentence we need) or to search for a set of words (basically, divide the sentence into words and get sentences in the TM with as many words from these as we can find — which does not guarantee order in the words).

To solve this problem we will need to prepare a specific indexer for the task. The semantics we want for this function is: to find a similar sentence where some words can differ, and some new ones can occur in the middle. For that purpose, we should define thresholds of how many words can differ and how many new words can be found.

Although with a simple definition, the implementation of such a function in an efficient way is not simple.

5.1.3 Perl implementation

Both Perl and PHP implementation are very straightforward. The version here presented is simplified and its objective is to illustrate how to build your own server.

Here we will show a Perl server. First, we need to create a module file. Name it `mtd.pm` for example. This module will have a function for each one of the web-service's methods[2].

```
package mtd;

use Search::Glimpse;
use strict;

our %known_languages = (
    en => [qw/pt/],
    pt => [qw/en/] );

sub lang_pair {
    my ($self, $l1, $l2) = @_;

    if (exists($known_languages{$l1}) &&
        grep { $_ eq $l2 }
           @{$known_languages{$l1}})
    { return 1 }
    else
    { return 0 }
}

sub equiv {
  my ($self, $l1, $l2, $string) = @_;
  my $x = Search::Glimpse->new(
        'server'  => 'server.url.pt',
        'nr_hits' => 10);
  my @answers  = $x->search($string);
  my $i = 0;
  for (@answers) {
    $i++;
    s!^[^:]+:!!;
    s/<tu.*<tu id="[^"]+">//;
    s!</tu.*$!!;
  }
  if ($answers[0] eq "ERROR")
    { return undef }
  else
    { return $answers[0] }
}

sub conc {
  my ($self, $l1, $l2, $string) = @_;
  my $x = Search::Glimpse->new(
        'server'  => 'server.url.pt',
        'nr_hits' => 20);
  my @answers  = $x->search($string);
  return undef if $answers[0] eq "ERROR";
  @answers = map { s!^[^:]+:!!;
                   s!^\s*<[^>]+>!!;
                   s!</tu>$!!;
                   s!</tu><tu[^>]+>!#XPTO#!;
                   [ split /#XPTO#/ ]
                 } @answers;
  return [@answers];
}
1;
```

[2]We know that the Perl code can be hard to read. Meanwhile, it is presented here to show how small an implementation can be.

As can be seen, using a different approach to index the translation memory will only change this module and the way it finds the answers.

5.2 Client Implementation

The client should not be an independent program but should be built-in or a plug-in for a CAT system. Meanwhile, for test purposes we developed two clients, in Perl and PHP. Both of them communicate with any of the two servers.

The implementation based on PHP and NuSOAP is also simple. Here follows how to query the server for the sentence "the pet is dead."[3]

```
<?php

  require_once('nusoap.php');
  $parameters = array('arg'=>'the pet is dead');
  $soapclient = new
       soapclient('http://server.url.pt');
  $answer =
    $soapclient->call('equiv',$parameters);

  foreach($answer as $v) {
    print "$v<br>";
  }
?>
```

6 Conclusions

The concept of Distributed Translation Memories is useful for translators communities to share their translations. It can also be used to sell or to rent translation memories.

The implementation using web-services has a big advantage over other RPC protocols: as it uses HTTP to communicate, messages can pass through firewalls and HTTP proxies without problems.

Although with some problems regarding translation memory search, the concept is prototyped and working. A WSDL file (Web-service description language) should be written so that other users can provide their services and/or clients.

Pending tasks include the incorporation of a client in open-source CAT systems avail-

[3]Of course that an implementation of DTM should be integrated with the translation software. This code example is just an example of how simple is the code needed to integrate DTM in the system.

able in the Internet (p.e. Frankenstein[4], OmegaT[5], ForeignDesk[6]).

Using the messages as described in this article, you can access our two servers at `http://sli.uvigo.es/CLUVI/mtd.php` and at `http://linguateca.di.uminho.pt/MTD`. The source-code for the clients and server can be found at `http://natura.di.uminho.pt/downloads/source/MTD`.

References

Abaitua, Joseba. 2001. Memorias de traducción en TMX compartidas por internet. *Revista Tradumàtica*, (0), octubre. `http://www.fti.uab.es/tradumatica/revista/num0/articles/jabaitua/imprim%ir.pdf`.

Christensen, Erik, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. 2001. Web services description language (wsdl) 1.1. `http://www.w3.org/TR/wsdl/`.

Gómez, Josu. 2001. Una guía al TMX. *Revista Tradumàtica*, (0), octubre. `http://www.fti.uab.es/tradumatica/revista/num0/articles/jgomez/imprimir%.pdf`.

Manber, Udi and Sun Wu. 1994. Glimpse: A tool to search through entire filesystems. Winter USENIX Technical Conference.

OSCAR. 2003. Open Standards for Container/Content Allowing Re-use — TMX home page. `http://www.lisa.org/tmx/`.

Savourel, Yves. 1997. TMX 1.4a Specification. Technical report, Localisation Industry Standards Association.

10 February 1998. *eXtended Markup Language (XML) version 1.0 recommendation*. World Wide Web Consortium. `http://www.w3.org/TR/1998/REC-xml-19980210.html/`.

[4] `http://www.sf.net/projects/frankenstein/`
[5] `http://www.sf.net/projects/omegat/`
[6] `http://www.sf.net/projects/foreigndesk/`