XML::TMX– Processamento de Memórias de Tradução de Grandes Dimensões

José João Almeida and Alberto Manuel Simões

Departamento de Informática
Universidade do Minho
{jj|ambs}@di.uminho.pt

Resumo As ferramentas de tradução assistida por computador tentam reutilizar as traduções realizadas pelo tradutor sempre que uma frase semelhante tenha sido já traduzida. Para o intercâmbio destes documentos foi definido um formato denominado TMX (Translation Memory Exchange) baseado em XML.
Este tipo de documento ganha facilmente tamanhos incomportáveis para o seu processamento com métodos tradicionais.
Neste artigo propomos uma metodologia de ordem superior para o processamento de documentos de estrutura repetitiva (em que se inserem as memórias de tradução) com uma abordagem baseada na conjunção de SAX e DOM.
São apresentados vários exemplos de filtros sobre memórias de tradução bem como um conjunto de medidas da sua eficiência.

1 Introdução

A tradução quando feita em ambiente profissional tira proveito de ferramentas de tradução assistida por computador (CAT). Estas ferramentas armazenam as traduções já realizadas em projectos anteriores em bases de dados específicas (memórias de tradução).

Dum modo simplificado as memórias de tradução são grande conjuntos de unidades traduzidas (designadas por unidades de tradução – normalmente frases) e opcionalmente algumas propriedades e anotações associadas.

1.1 Processo de Tradução

A tradução baseada em memórias de tradução tem como principal objectivo a reutilização de traduções já realizadas pelo tradutor. Uma memória de tradução não é mais do que uma base de dados de segmentos traduzidos (unidades de tradução) que permitem ao tradutor[4,6]:

- propagar no texto de destino as traduções de frases que se repetem no texto original;
- reciclar traduções que foram realizadas ao trabalhar noutros textos, reutilizando-as tal como armazenadas na memória de tradução ou depois de alteradas;

– analisar um novo texto original e encontrar *matches* com traduções já armazenadas na memória de tradução, permitindo desta forma reutilizar porções de traduções já realizadas;

O processo de tradução usando memórias de tradução realiza-se da seguinte forma:

1. o programa divide o texto original em segmentos. Esta divisão é feita tendo em conta a pontuação da língua em causa, e a marcação do formato específico em que o documento se encontra;
2. a tradução é realizada para cada segmento do texto de origem pela sua ordem natural, de acordo com os seguintes passos:
   (a) o programa verifica se o próximo segmento a ser traduzido está na memória de tradução, ou se algum segmento razoavelmente semelhante já foi traduzido;
   (b) o tradutor determina se vai usar, editar ou ignorar a tradução que o programa encontrou;
   (c) o programa guarda o segmento da língua de origem e a respectiva tradução na memória de tradução;

São exemplos de ferramentas de tradução assistida por computador o Trados [9], STAR Transit [13], Déjà vu [2] e mesmo o sistema colaborativo de tradução TransBey [3].

1.2 Memórias de Tradução: TMX

O formato TMX [7] foi definido pela associação LISA (The Localization Industry Standards Association) como forma de partilhar memórias de tradução entre sistemas de tradução (que usam internamente formatos proprietários).

Sendo um formato de intercâmbio, é fortemente estruturado. No entanto, bastante simples. É composto por duas partes principais: um cabeçalho de meta informação, e um corpo. Esta segunda parte é a principal destes documentos, composta por pequenas entradas de unidades de tradução.

Segue-se um pequeno exemplo:

```
1   <?xml version='1.0' encoding='ISO-8859-1'?>
2  <!DOCTYPE tmx SYSTEM "tmx11.dtd">
3  <tmx version="version 1.0">
4  <header
5           creationtool="cwb-utils"
6           creationtoolversion="1.0"
7           segtype="sentence"
8           adminlang="EN-US"
9           srclang="fr"
10          o-tmf="CQP-corpora"
11 >
12 </header>
13 <body>
```

```
<tu>
 <tuv lang='pt'>
    <seg>Praticamente ausente dos mapas de fluxo de dados, a África
    não contabiliza mais linhas telefónicas do que Tóquio ou Manhattan,
    nem mais computadores ligados à Internet do que a Lituânia.</seg>
 </tuv>
 <tuv lang='fr'>
    <seg>Quasi absente des cartes de flux de données, l'Afrique ne
    compte pas plus de lignes téléphoniques que Tokyo ou Manhattan, pas
    plus d'ordinateurs connectés à Internet que la Lituanie.</seg>
 </tuv>
</tu>

<tu>
 <tuv lang='pt'>
    <seg>Todavia, o continente não escapa às transformações nas
    telecomunicações, onde se lêem, mais do que em qualquer outro sítio,
    as recomposições inéditas impostas pela mundialização.</seg>
 </tuv>
 <tuv lang='fr'>
    <seg>Pourtant, le continent n'échappe pas au bouleversement des
    télécommunications, dans lequel se donnent à lire, là plus
    qu'ailleurs, les recompositions inédites qu'impose la
    mondialisation.</seg>
 </tuv>
</tu>

</body>
</tmx>
```

2 Processamento de TMX de Grandes Dimensões

Embora seja um formato bastante simples, é muito habitual os documentos TMX chegarem às centenas de MegaBytes. Deste modo, é praticamente impossível processar estes documentos em memória da forma convencional. Uma árvore do DOM (Document Object Model) de um destes documentos ocupa cerca de 15% mais de memória do que o documento original só em estruturas de dados.

Por outro lado, o outro modelo de processamento habitual baseado em SAX não é suficientemente versátil.

Surge então a necessidade de analisar uma metodologia que nos permita por um lado processar documentos de grandes dimensões, e por outro, garantir alguma versatilidade. Para isso decidiu-se processar o corpo da TMX de uma forma incremental, por blocos. Cada memória de tradução pode ser processada independentemente da outra. Por outro lado, cada memória de tradução é um documento XML válido por si só, o que permite que se use um parser para criar o DOM desse bloco.

A implementação deste algoritmo no nosso módulo XML::TMX[10] é bastante simples. A linguagem Perl permite a definição de um separador de registo, separador esse que é usado pelos métodos de leitura de ficheiros (ou *standard input*), para ler porções (ou *chunks*) de informação. Desta forma, é possível definir o separador de registo como sendo a etiqueta de fecho de uma unidade de tradução (`</tu>`) o que permite que cada chunk (com excepção do primeiro e do último) seja uma unidade de tradução completa.

```
<tu>
 <tuv lang='pt'>
    <seg>Praticamente ausente dos mapas de fluxo de dados, a África
    não contabiliza mais linhas telefónicas do que Tóquio ou Manhattan,
    nem mais computadores ligados à Internet do que a Lituânia.</seg>
 </tuv>
 <tuv lang='fr'>
    <seg>Quasi absente des cartes de flux de données, l'Afrique ne
    compte pas plus de lignes téléphoniques que Tokyo ou Manhattan, pas
    plus d'ordinateurs connectés à Internet que la Lituanie.</seg>
 </tuv>
</tu>
<tu>
```

Cada um destes extractos é por si só um documento XML válido (com a excepção da definição de XML) que pode ser interpretado usando o XML::DT [1].

Embora esta abordagem obrigue à inicialização de um *parser* XML para cada uma das unidades de tradução, é escalável: não é necessária a criação da árvore de DOM em memória.

Esta abordagem, embora neste caso específica para as memórias de tradução é facilmente generalizável para outros tipos de documentos como sejam os de anotação de terminologia dicionários TEI e SKOS, subconjuntos de OWL, e outros.

3 Processamento de ordem Superior

Esta secção inicia com uma introdução da abordagem na programação de sistemas de parsing de ordem superior [5], seguida de um conjunto de exemplos da utilidade desta metodologia de programação.

3.1 Abordagem

Dado que o processamento das memórias de tradução não é eficiente como um todo, e por outro lado, ao processar cada unidade de tradução é indiferente a sua vizinhança (não há noção de ordem de memórias de tradução), é possível definir uma função de processamento de memórias de tradução que receba como parâmetro a função que processa cada memória de tradução.

Para além do resultado da função de processamento da unidade de tradução em causa, a função de ordem superior **for_tu**, permite:

- **transformar as unidades de tradução:** $tu \rightarrow tu|0$
  a função de processamento **for_tu** funciona como um *map* funcional que aplica a cada unidade de tradução uma função de processamento que pode devolver a unidade de tradução depois de processada ou produzir efeitos laterais.
- **remover unidades de tradução:**
  no caso da função de processamento devolver um objecto vazio, a unidade é retirada da memória gerada.
- **adicionar e remover propriedades às unidades de tradução:**
  além do texto e respectiva tradução o *standard* TMX permite definir propriedades (etiqueta `prop`) e notas (etiqueta `note`) sobre cada unidade de tradução. A função de processamento recebe não só o texto correspondente à unidade de tradução mas também a lista de propriedades e de notas associadas. A função de processamento permite adicionar e remover propriedades e notas.
- **indicar o ficheiro de saída pretendido:**
  por omissão a função escreve a nova memória de tradução para o *standard output*. No entanto este comportamento pode ser alterado indicando o nome do ficheiro onde a nova memória deve ser escrita.
- **definir um número máximo de unidades de tradução a processar:**
  em algumas ferramentas, como as que funcionam sobre a web, é importante limitar o número de unidades de tradução a processar de forma a aliviar o processamento. Este número pode ser definido ao invocar a função **for_tu**.
- **definir o número máximo de unidades de tradução a escrever:**
  funciona de forma semelhante à anterior, mas em vez de limitar o número de unidades de tradução a processar limita o número de resultados: processa unidades de tradução até que seja retornado o número de unidades de tradução escolhido.
- **indicar um padrão de activação:**
  permite especificar uma expressão regular de pesquisa, de forma a que apenas as unidades de tradução que façam *matching* sejam executadas.

3.2 Leitura de TMX: contar as TUs

Este exemplo trivial tenciona mostrar o quão simples é escrever um processador de memórias de tradução. Neste processador faz-se uma travessia de uma memória de tradução com a finalidade de contar o número de unidades de tradução existentes.

```
1  use  XML::TMX::Reader;
2  my $mem = XML::TMX::Reader->new('sample.tmx');

3  my $count = 0;
4  $mem->for_tu(
5      sub { $count++; }
6    );

7  print $count;
```

linha 1: é carregado o módulo para leitura de TMX;
linha 2: é criado um objecto com a TMX em causa;
linha 4: o método **for_tu** itera todas as memórias de tradução;
linha 5: a função de processamento da memória de tradução limita-se a contar a existência de mais uma ocorrência.

3.3 Transformação de Memórias de Tradução

Considere-se o seguinte processador genérico de memórias de tradução:

```
1  my $mem->for_tu(
2     { output => "nova.tmx",
3       gen_tu => 1000 },
4      \&proc )
```

linha 1: iterar as unidades de tradução;
linha 2: colocar o resultado do processamento no ficheiro "nova.tmx";
linha 3: o processamento termina após produzir 1000 unidades de tradução;
linha 4: a função de processamento a utilizar chama-se **proc**.

Seguem-se várias funções de processamento (**proc**) para o tratamento de unidades de tradução.

Remoção de unidades de tradução Ao criar memórias de tradução automaticamente é habitual existirem maus alinhamentos (unidades de tradução cujo texto não corresponde à tradução correcta).

Se uma unidade de tradução tiver segmentos com tamanhos muito díspares é do nosso interesse removê-la. Neste exemplo considera-se que as línguas em causa são a língua inglesa (en) e a portuguesa (pt).

```
1     sub proc {
2       my $tu = shift;
3       if (length($tu->{en}) > 2 * length($tu->{pt})
4            { return undef }
5       else { return $tu ; }
6     }
```

linha 2: colocar em `$tu` a unidade de tradução;
linha 3: calcular o tamanho do segmento inglês e português (acedidos com `$tu->{en}` e `$tu->{pt}`) e devolver indefinido se o primeiro tiver mais de duas vezes o tamanho do segundo;
linha 5: devolver a unidade de tradução para ser adicionada ao resultado gerado.

Adicionar propriedades Como foi referido anteriormente, cada unidade de tradução pode ter a si associado um conjunto de propriedades e notas. A seguinte função de processamento adiciona uma medida de qualidade de tradução e um domínio.

```
1    sub proc {
2      my $tu = shift;
3      $tu->{-prop}{quality} = calculate_quality($tu->{en},$tu->{pt});
4      $tu->{-prop}{domain}  = infer_domain($tu->{pt});
5      return $tu;
6    }
```

3.4 Limpeza de TMXs

Em todo os trabalhos envolvendo extracção de informação a partir de TMX há naturalmente uma grande dependência da qualidade das unidades de tradução da TMX de partida. Neste exemplo a estratégia usada passa pela contagem e comparação de elementos não textuais (ex, números) que permanecem iguais nas duas línguas: serão rejeitadas as unidades que difiram muito nos elementos não textuais, bem como as que tenham tamanhos excessivamente diferentes. Por último, são rejeitadas as memórias de tradução (inteiras) quando a taxa de unidades de tradução rejeitadas for elevada.

```
1  Para todas a TMX f fazer:
2     perfeitas = 0                 # contador de TU muito boas
3     péssimas = 0                  # contador de TU muito más
4     numbers_pt = {}               # números na parte portuguesa
5     numbers_en = {};              # números na parte inglesa

6     nova.tmx = limpa(f.tmx)
7     se (numbers_pt ≠ numbers_en)
8         apaga(nova.tmx)
9     se (taxa(péssimas) é elevada  )
10        apaga(nova.tmx)
```

onde `novaf.tmx = limpa(f.tmx)` corresponde a:

```
1  $mem->for_tu(
2    {output => "novaf.tmx"} ,
3    \&proctu
4  )
```

sendo a função de processamento das TUs `proctu` acima referida, definida como:

```
1  sub proctu {
2    my $tu = shift;
3    $tu->{pt} = retoca($tu->{pt});
4    $tu->{en} = retoca($tu->{en});
5    ...extrai números pt en
6    ...perfeita  se   números pt = números en ≠ {}
7    ...péssima   se   números pt muito diferentes números en
8    ...péssima   se   tamanhos muito diferentes
9    confiança = f(....)
10   if (péssima) { return undef }
11   $tu->{prop}{conf} = confiança
12   return $tu;
13 }
```

3.5 Remoção de entradas duplicadas

Por vezes ao criar e juntar memórias de tradução acabam por existir unidades de tradução repetidas. Este exemplo mostra uma forma rápida de as remover, usando para isso o valor MD5 da unidade de tradução.

```
1  tie %dic, 'DB_File', "mydbfile.db",
2      O_RDWR|O_CREAT|O_TRUNC , 0640, $DB_BTREE;
3  my $tm = XML::TMX::Reader->new($filename);
4  $tm->for_tu(
5      sub {
6          my $tu = shift;
7          my $digest = md5_hex("$tu->{en},$tu->{pt}");
8          if ($dic{$digest}) {
9                 return undef
10         } else {
11                $dic{$digest} = 1;
12                return {%$tu} ;
13         }
14     }
15 );
```

linha 1: criar uma base de dados de valores MD5 para consulta rápida;
linha 5: iterar todas as memórias de tradução;
linha 7: calcular o valor MD5 da memória de tradução;

linha 8: se o valor MD5 está na base de dados, a unidade é repetida, e portanto, ignorar;

linha 10: se o valor não existe, guardar na base de dados e devolver a unidade de tradução.

4 Considerações referentes a desempenho

A tabela 1 mostra uma comparação de tempos[1] da função **for_tu** implementada com base na construção da árvore DOM e com base no processamento por chunks. Foi construída uma função que apenas conta o número de unidades de tradução (ver secção 3.2) e aplicada a memórias de tradução com diferentes tamanhos.

Note-se que enquanto é possível guardar todo o DOM em memória esta abordagem é mais eficiente. No entanto, assim que deixa de caber em memória passa a não ser exequível (no nosso ambiente de teste a memória de tradução EurLex congela o computador).

Por outro lado, a abordagem por chunks tem um crescimento linear. Embora possa demorar mais tempo consegue dar uma resposta. Note-se que é normal utilizar memórias de tradução com mais de um milhão de unidades de tradução.

			DOM		Chunks	
	TUs	Tamanho	tempo	memória	tempo	memória
NATO	53 562	18 MB	38s	108 MB	50s	10 MB
LMD	68 231	25 MB	41s	145 MB	61s	10 MB
PE1	382 581	83 MB	230s	637 MB	343s	10 MB
EurLex	1 110 163	353 MB	—	—	1003s	10 MB

Tabela 1. Tempos de parsing simples.

Considerando a medida de um exemplo mais complexo, a remoção de unidades de tradução repetidas (ver secção 3.5) numa memória de tradução com 1 784 164 unidades (560MB) demora cerca de 35 minutos e 25 segundos (removendo 714 837 unidades repetidas).

5 Conclusões

As memórias de tradução são recursos bastante úteis, quer para tradutores, linguistas ou investigadores de processamento de linguagem natural (por exemplo, na tradução automática [8]). Deste modo, é importante que existam métodos eficientes para o seu processamento. Embora existam ferramentas que utilizam estas memórias de tradução para ajudar o tradutor no seu trabalho, elas tratam

[1] As medidas apresentadas nesta secção foram obtidas num Pentium IV 3.2GHz, com 2GB de RAM, Linux.

apenas desta utilização específica das memórias de tradução. Há uma grande necessidade e utilidade em dispor de um ambiente de programação versátil e aberto para o tratamento generalizado de memórias de tradução.

A resolução do problema da disponibilização eficiente de memórias de tradução já foi discutida em [12,11], utilizando uma arquitectura distribuída de servidores dedicados. No entanto, esta abordagem não facilita o tratamento preliminar das memórias de tradução.

A abordagem apresentada tem a vantagem de ser escalável a memórias de tradução de grandes dimensões, sem que o seu tamanho influencie a memória necessária para o seu processamento: o tempo cresce linearmente em relação ao número de unidades de tradução.

O facto de se ter optado pelo uso de funções de ordem superior permite que com pouco esforço de programação se façam filtros ou apenas travessias a memórias de tradução. A própria facilidade do uso de módulos externos (como o exemplo da secção 3.5) é crucial no desenvolvimento de ferramentas.

Acreditamos que a metodologia mista de uso SAX (escalabilidade) e DOM (expressividade) é aplicável a uma grande variedade de outro tipos de documentos. Esta abordagem só traz resultados vantajosos quando os documentos têm estrutura repetitiva (nas outras situações funciona mas não tem vantagens) o que acontece na generalidade do uso de XML em documentos de grandes dimensões.

Agradecimentos

Este trabalho foi parcialmente financiado pela Fundação para a Ciência e Tecnologia of Portugal pela bolsa POSI/PLP/43931/2001, e co-financiado pelo POSI, dentro da Linguateca.

Referências

1. J.J. Almeida and José Carlos Ramalho. XML::DT a perl down-translation module. In *XML-Europe'99, Granada - Espanha*, May 1999.
2. ATRIL. Déjà vu home page and documentation. Translation tool, October 2006. `http://www.atril.com/`.
3. Youcef Bey, Christian Boitet, and Kyo Kageura. The TRANSBey prototype: an online collaborative wiki-based cat environment for volunteer translators. In *LREC-2006: Fifth International Conference on Language Resources and Evaluation. Third International Workshop on Language Resources for Translation Work, Research & Training (LR4Trans-III)*, pages 49–54, Genoa, Italy, 28 May 2006.
4. Claude Bédard. Mémoire de traduction cherche traducteur de phrases (translation memory is looking for sentences translator). *Traduire (Traduire) ISSN 0395-773X*, (186):41–49, 2000.
5. Mark Jason Dominus. *Higher Order Perl*. Morgan Kaufman, 2005.
6. Dorothy Kenny. Translation memories and parallel corpora: Challenges for the translation trainer. In *Inaugural Conference of the International Association for Translation and Intercultural Studies*, Sookmyung Women's University, Seoul, Korea, 12–14 August 2004.

7. Yves Savourel. TMX 1.4a Specification. Technical report, Localisation Industry Standards Association, 1997.
8. Reinhardt Schaeler. Beyond translation memories. In Michael Carl and Andy Way, editors, *Workshop on Example-Based Machine Translation*, pages 49–55, September 2001.
9. SDL International. Sdl trados home page and documentation. Translation tool, October 2006. `http://www.trados.com/`.
10. Paulo Silva, Alberto Simões, and José João Almeida. XML::TMX – Perl extensions for managing TMX files. Perl module, Projecto Natura, Departamento de Informática, Universidade do Minho, 2003. `http://search.cpan.org/dist/XML-TMX-0.14/`.
11. Alberto Simões, José João Almeida, and Xavier Gomez Guinovart. Memórias de tradução distribuídas. In José Carlos Ramalho and Alberto Simões, editors, *XATA2004 - XML, Aplicações e Tecnologias Associadas*, pages 59–68, February 2004.
12. Alberto Simões, Xavier Gómez Guinovart, and José João Almeida. Distributed translation memories implementation using webservices. In *Sociedade Española para el Procesamiento del Lenguaje Natural*, pages 89–94, Jul. 2004.
13. STAR Group. Transit home page and documentation. Translation tool, October 2006. `http://www.star-group.net/`.