Activity Report at SYSTRAN S.A.

Pierre Senellart

September 2003 – September 2004

1 Introduction

I present here the work I have done as a software engineer with SYSTRAN. SYSTRAN is a leading company in machine translation, with more than 40 supported language pairs, and is providing most of the online machine translation service (Google, Yahoo, Altavista...). SYSTRAN regroup SYSTRAN S.A. (Paris, France) and SYSTRAN Software Incorporated (San Diego, USA). I mostly worked in Paris but stayed 4 weeks in San Diego, for the conclusion of a corporate customer project and the finalization of SYSTRAN version 5 for East Asian languages. My work focused on the development and finalization of version 5 commercial products, which include R&D on new features (cf Section 2), engineering tasks on SYSTRAN architecture (cf Section 3) and a number of other smaller tasks, such as bug fixing, maintenance, code cleaning and *ad hoc* coding. The development was mostly done with Perl, C++ and XSLT, under Linux and Microsoft Windows. I am either the sole author or a main contributor of the works presented here, under the direction of Jean Senellart, Director R&D of the SYSTRAN group.

2 Research and Development

2.1 Translation memories

A translation memory is a set of sentences in a source language stored with their translation in one or several languages. Translated sentences may or may not include character properties (e.g bold, italic. . .) in addition to the text. Translation memories are often used by (human) translators and translation agencies to store the translation results and for the reuse of translations. TRADOS is the most used commercial translation memory software; TMX (Translation Memory eXchange format) is a standard XML format for describing translation memories.

Translation Memory Support (with full support of character properties) was added to SYSTRAN v5, with the possibility of compiling and using translation memories, and the possibility of generating or completing translation memories with machine translation. The TMX format is also supported.

2.2 Entity translation

As SYSTRAN linguistics code (originally written in an assembly language and then converted to C) is quite complex, the possibility of short-circuiting it (while still interacting with it) for some new, independent, linguistics development is interesting. Entity translation is about the recognition of common linguistics entities in the source text (e.g. dates, address, numbers written with letters...) and its translation to the target language, while providing information to the linguistics routine to deal with this entity in a proper way. For instance, if “on March 23rd” is recognized by the entity analysis module as a date complement, it should be considered as an adverb in the linguistics routines. These concepts allowed in particular an important improvement in the quality of date translation (e.g. in English to French, “March 23rd” was formerly translated as “Mars 23rd” and is now correctly translated as “(le) 23 mars”).

2.3 SYSTRAN Translation Stylesheets

XSL Transformation stylesheets are usually used either for transforming a document described in an XML formalism into another XML formalism, for modifying an XML document, or for publishing content stored in an XML document to a publication format (XSL-FO, (X)HTML...). SYSTRAN Translation Stylesheets (STS) use XSLT to drive and control the machine translation of XML documents (native XML document formats or XML representations (as XLIFF) of other kind of document formats).

XSLT does not only provide a simple way to indicate what part of the document text is to be translated, but also allows for fine-tuning of translation, especially by using the structure of the document to help disambiguate natural language semantics and determine proper context. For instance, the phrase “Access From Front Door” is to be analyzed as “The access from front door” within a title, and as “Do access (something) from front door” in text body. In that case, the STS would pass ‘title’ option to the translation engine. In the same way, the stylesheet can activate specialized domain dictionary for some part or other of the document and can mark some expressions not to be translated.

Another key application of STS is to consider machine translation as part of the authoring and publication process: source documents can be annotated with natural language markup produced by author which will be processed by STS to improve the quality of translation, leading, for instance, to the automatic publication of a multilingual website from a monolingual (annotated) source. This positioning inside the publication process is a real break-through for web content translation. Traditionally, machine translation applies after publication and do not have access to original structure of the document but only to its HTML representation.

The mechanism is implemented through XSLT extension functions. In particular, the stylesheet uses a ‘translate’ function to translate an XML fragment, and get/push/pop functions for consulting and setting linguistics options in the translator. A way for proper management of character properties is also provided so that, for instance, the translation of a phrase in bold font will be in bold font, even if the phrase has moved in the translated sentence.

This process is highly customizable by the addition of new templates into the stylesheets. Because the translation is driven by the document structure, it is much easier to use this structure during translation and keep this structure in the translated document, than with traditional document filters where the entire document is processed linearly.

STS (already partially implemented in v4) were implemented in v5 and stylesheets were written for:

- Generic XML formats
- XHTML and HTML
- TMX
- XLIFF

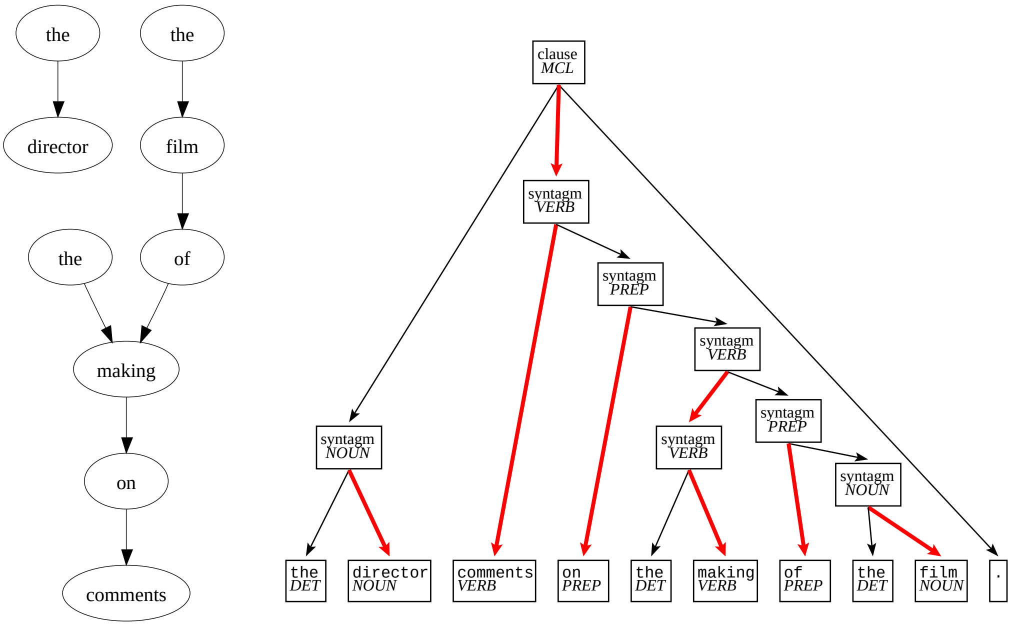

Figure 1: On the left, graph of the relations between words in the sample sentence "The director comments on the making of the film." (the relation types are not shown). On the right, syntagm tree for the same sentence, derived from the graph; head words of each syntagm are linked to the syntagm with a bold edge.

2.4 Syntactic DAG building from word-to-word relations

SYSTRAN's classical translators (i.e. non-New Generation translators) use a representation of the syntax of a sentence based on a set of binary relations between words (especially *modifier* relations). For instance, an article is linked to its determining noun, the head word of a prepositional phrase is linked to the preposition, etc. This can be represented as a directed graph, as shown in Figure 1 (left). This representation is not always convenient to handle in transfer and synthesis; for instance, it can be cumbersome to move an entire noun phrase to another position when synthesizing the target sentence. Moreover, a classical tree-like view of the syntax analysis is easier to read by developers, linguists and could even be useful for the final user. Therefore, SYSTRAN translators now integrate a routine which convert the classical representation into a tree of syntagms — more precisely, it is a directed acyclic graph (DAG) in case of a complex set of syntactic relations. Figure 2 shows the corresponding algorithm.

Another advantage of the tree-based representation is that it is closer to the representation of the analysis of other tools (including SYSTRAN New Generation systems). The algorithm detailed in Figure 2 can be reversed to derive a set of relations between words from a syntax tree. The corresponding relations can then be used as usually by classical transfer and synthesis modules. This is another step into modularization since the analysis module can be replaced by the output of another tool which produce syntax trees. Another application is the possibility for the user to act on the result of the analysis, and to re-inject the modified syntax tree into the translation engine.

```
SyntaxGraphToSyntagmDAG
Input: a graph G of labelled syntactic relations (acyclic)
Output: a DAG T of syntagms

For each connected component H in G:
  1. Build a mapping M which maps each word node to an elementary
     syntagm containing this word.
  2. Select a node n from H with no incoming link.
  3. For each n' in relation with n (order by relation kind priority)
     (a) Recursively call step 2 with n' as new n, which gives a syntagm DAG T'
     (b) Create a new syntagm s, with M(n) as head word and T' as
         other constituent. Put the result in M(n) and return it.
(Connected components are finally regrouped into clauses, and relations
between clauses are computed. Enumerations are also handled, this is
not detailed here.)
```

Figure 2: Algorithm for converting a syntactic relation graph to a syntagm DAG

3 Engineering

3.1 UTF-8 conversion

SYSTRAN translation engines used special ASCII transliterations, dependent of the language, to represent most source and target texts. For instance, ê was written as `2e` in most European languages, but as `'e` in Spanish source documents. This caused a large number of potential problems in translations:

- Real occurrences of `2e` had to be protected in the text. A protection mechanism existed (`2e` was written `2$e`) but was far from perfect (it was not dealt with in all circumstances, real occurrences of `2e` were not correctly protected, etc.)
- Representation of characters which had no transliterations in a particular language was quite problematic. This could provoke, for instance, foreign words to disappear in the translation; when specific mechanisms existed (e.g. in Greek or Russian for represented Latin words), the mechanism introduced other issues by adding other kind of special characters.
- Linguistics routines did not always correctly manage these special characters: a routine which had to remove the last character of a word may for example only remove the `e` in `2e`.
- There were sometimes confusions in the transliterations to use (in Spanish and German, the transliterations were not the same for source and target texts).
- Some transliterations were not reversible (for instance, in Russian, the case of some characters was not preserved in the transliteration).

The way to solve all these issues was to make the translation engines work directly on UTF-8 characters. This means:

- Converting all dictionaries to UTF-8.
- Converting all hard-coded strings to UTF-8 in the source code. This was done by a first automatic pass and a manual review of every modified source files (several thousand of source files!).
- Adapting all linguistics routines directly working on characters (e.g. elision, special inflexions...).
- Removing all remaining references to transliteration.

3.2 Compilation options

Compilation options were originally added either to the `make` command line or to a configuration file, included in each source file. There was no way to know how a particular binary translator was compiled, nor to manage complex dependencies between compilation options. To fix these issues, a new *configuration* mechanism was introduced, through the means of a Perl `configure` script which parsed an XML description of all available compilation options and of all compilation flavors (groupings of options). Dependencies between options were described by implies/needs/conflicts entries, while runtime querying of the compilation options was made possible.

3.3 Runtime options

Translation engines used a number of runtime input and output options, for linguistics preferences (should English "you" be translated by "tu" or "vous" in French?), for feature activation (address recognition) or for a vast number of other needs. This was implemented with a simple C++ map and worked well. The problem was that the engine did not have any knowledge about the supported options. The user, and every portion of the code, could query or set any option name, which sometimes did not have any sense; typos were numerous too. Moreover, there was no central documentation of supported options.

I implemented what follows. Runtime options are described in an XML file, with their name, natural language description, type (string, number. . .), input/output capabilities. . . Each access to the option map is then checked against this description and errors issue a runtime warning. HTML or TXT descriptions of supported options are also provided, through XSL stylesheets. Furthermore, because of the plug-ins described in the next section, runtime option descriptions can also be added at runtime.

3.4 Dynamic loading of document filters

SYSTRAN translators have to be compiled for each different language pair. This is quite a long process, and an error anywhere in the code implies a complete recompilation of all translators. This is not acceptable and an effort for a better modularization had to be done. The first step was to isolate a number of document filters (in a first time, PDF and Microsoft Word filters) from the translation engines, in the form of dynamically loaded modules (DLLs, using Microsoft Windows terminology). An oriented-object mechanism for declaring filter modules have been implemented and the best available filter is selected at runtime.