Integration of SYSTRAN MT systems in an open workflow

Mats Attnäs
SYSTRAN S.A.
La Grande Arche
1, Parvis de la Défense
92044 Paris La Défense
France
mattnas@systran.fr

Pierre Senellart
SYSTRAN S.A.
& INRIA Futurs
Parc Club Orsay Université
4 rue Jacques Monod - Bât G
91893 Orsay Cedex France
psenellart@systran.fr

Jean Senellart
SYSTRAN S.A.
La Grande Arche
1, Parvis de la Défense
92044 Paris La Défense
France
jsenellart@systran.fr

Abstract

A general mature rule-based MT system is bound to reach a saturation point because of the intrinsic complexity of the natural language description. For such systems, maintenance is a complex task and customization is expensive and time-consuming. Therefore, improving the system's interaction with the linguistic rules has proven to be more productive than endlessly adding new rules and exceptions to reach the theoretical accuracy level of 100%. In this paper, we describe our strategy to "open up" such a system and provide practical details on how this was done on SYSTRAN's classical engines. This approach is the foundation of the SYSTRAN version 5 translation engines. We show the immediate benefits of the evolution of our architecture and the new extension potentiality.

1 Introduction - background

Over the last 30 years, SYSTRAN has developed and brought to a commercial level about 40 language pairs plus more than 20 other language pairs still in development or limited to a specific usage. These systems covers 20 different languages for which corresponding analysis or synthesis has been incrementally built, which represents several hundred thousands of "linguistic rules". This huge linguistic asset has evolved along with programming languages and is now an aggregate of several generations of rules and rule exceptions.

This code makes a complex set from which it is impossible to extract a "clean" formalized set of rules and which is difficult to conceptualize. Part of this complexity is bound to language complexity requiring that each rule comes with its own set of exceptions, another part comes from the fact that assembling and ordering a large number of linguistic rules is intricate whatever the formalism these rules are represented in. Additionally any modification to this complex set of rules systematically introduce risks of degradations.

Therefore, maintenance and evolution of these 60 classical translation engines has become a challenge all the more critical as last years have seen the emergence of a growing demand for highly customized MT solutions, that is the need for adapting and fine-tuning a translation engine linguistic rules for a specific domain or application. The same problematic is reported in (Keh-Yih Su, 1999).

At the same time, an important effort to build new rule-based systems based on state-of-the-art NLP development concepts has been carried out (J. Senellart, 2001). First commercial systems (Arabic→English, and Swedish→English) belonging to these "SYSTRAN New Generation" (NG) engines have been released in 2004 as part of SYSTRAN version 5.

In this article, we describe the strategy chosen for tackling this challenge. This strategy is based on the following idea: the solution is not to maintain the existing systems as they are, or to rebuild systems from scratch on a clean basis (not thinkable considering manpower invested in the existing systems), and not to consider them as black boxes that would seriously limit possibility of improvement — but to leverage existing systems by:

- modernizing the general architecture
- defining a modular workflow for the translation process (J. Senellart, 2003)
- introducing control mechanisms within the workflow, and within linguistic modules
- merging system maintenance with NG development, especially allowing for system extension in a declarative manner on high level linguistic objects

In section 2, we describe the key concepts of this architecture evolution. In section 3, we illustrate how this architecture allows to improve translation quality in a productive way. We conclude in section 4 by giving new perspectives and additional evolution axis.

The architecture described in this paper is the basis of all SYSTRAN version 5 systems released in June 2004.

2 New Architecture - Key concepts

We describe in this section the key concepts of the new architecture built to facilitate maintenance of classical systems and to get the maximum from their linguistic rules. This include code modernization, integration within an "XML workflow", introduction of high level data structures and a powerful lookup mechanism. Figure 1 describes this global architecture.

2.1 Code modernization

SYSTRAN translation engines were originally written (in the 70s) in an assembly language; the code was then migrated to the C programming language. Following the evolution of programming languages and software design, more and more portions of the code are now written in C++. More than a simple rewriting, the switch to an object-oriented language implies:

- a real redesign of the corresponding layers of the engines, simplifying modularization tasks as described in section 2.2
- the introduction of high level linguistic data structures as described in section 2.3

Another recent evolution of software comes with the introduction of the Unicode standard (The Unicode Consortium, 2003), which provides a way to represent in a single character repertoire all characters needed for most of the natural languages alphabets, along with character encodings (namely UTF-8, UTF-16 and UTF-32) representing these characters as sequence of bytes. This is a real change from traditional character encodings (ISO-8859-1, SHIFT-JIS, BIG5...) which did not permit, for instance, the representation in the same document of French and Chinese texts. SYSTRAN translation engines, while supporting a variety of character sets in the source documents, internally used a combination of traditional encodings and ASCII transliterations. This caused a number of limitation and difficulties, partly because workarounds had to be designed to represent different alphabets in the translation flow and partly because transliterations were intricate to manage inside linguistic routines. To solve these issues, SYSTRAN translation engines have been made compliant to the Unicode standard, using UTF-8 as their internal character encoding.

2.2 Process Modularization and XML workflow

2.2.1 XML workflow

The SYSTRAN translation process is organized as a workflow of modules working on an XML document which is a representation of the input text, along with all linguistic and structural information added by preceding modules. Each module, from the source document filter to the target document filter, through all other steps, can read (*get*) this information and add (*post*) new information to the XML document. This information regroups, between other:

- the input text
- structural information about the document format (HTML, Microsoft Word...), needed to rebuild the document
- typographical information (*character properties*) about the input text (e.g. which words were in bold/italic font), needed to put the same typographical properties on the translated text
- the translated text, aligned with the input text
- markup added to communicate with the other modules and with external applications (such as SYSTRAN Translation Project Manager): which words were spell-checked, what dictionary translated words come from, etc.

Section 2.3.1 details the way this information is represented as XML.

2.2.2 Modules

The XML workflow described in the previous section makes it possible to have a better modularization of the code: since the inputs and outputs of each XML workflow module are completely defined by their XML representation, each module is technically separable from the rest of the code (as an external library for instance, or even as a webservice). The XML workflow can thus be seen as a flow of XML, successively processed by a number of different independent agents, which get and post information to the workflow. Such a separation has a vast number of advantages: easiness to add

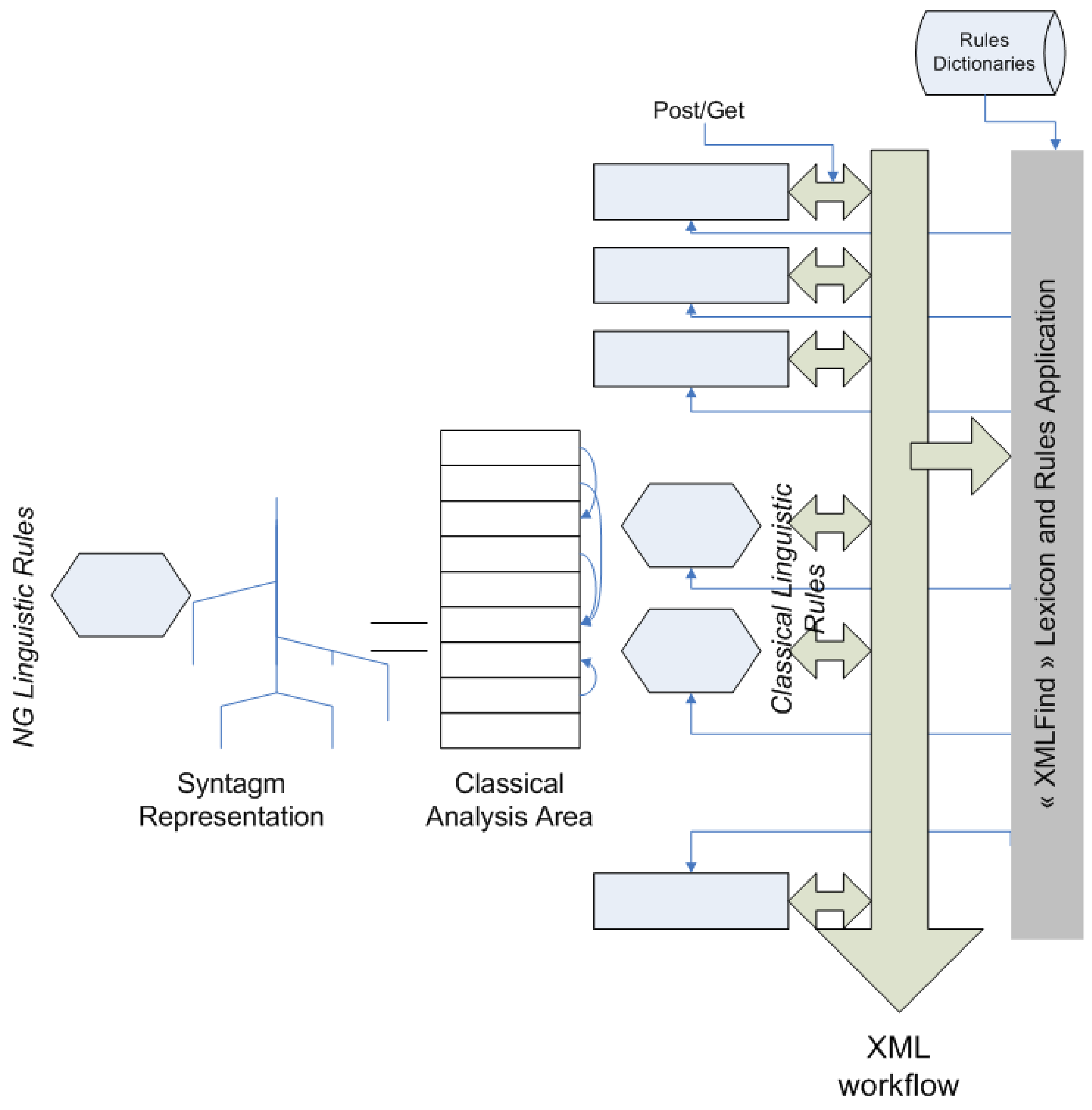


Figure 1: This schema describes the translation process. Boxes are the different modules. Modules communicate together using a get/post mechanism and publish their output using the "XML workflow". The classical analysis area is synchronized with a "NG" syntagm representation on which "NG" rules apply. Each module use a generic lookup mechanism (*XmlFind*, rightmost box) to recognize patterns or trigger rules on the current sentence representation.

or remove modules, possibility to split modules across machines for load balancing, more maintainable code...

Modules can be split into two different kinds: kernel modules, which are independent of the source and target languages (filters, preprocessing, normalization...), and linguistic modules (analysis, transfer, synthesis and other language dependent routines). It is to be noted that the isolation of linguistic modules accelerates the development of new language pairs, since the portion of the code dependent of both the source and target languages (that is, the transfer module) is clearly identified as the main point to work on. Work is in progress on the development of a generic transfer module, which would directly give a first version of a translation engine for new language pairs, using existing analysis and synthesis routines.

2.3 Data structures

2.3.1 The XML structure of the XML workflow

The XML workflow structure contains the current state of the analysis for a given sentence, as well as "history" information on the sentence evolution (allowing to potentially “undo” an action performed by a previous module), and generic “markup” used for communication between the different modules. Main features of this structure are described in (J. Senellart, 2001).

2.3.2 From a relation graph to a syntagm DAG

SYSTRAN’s classical engines (i.e. non-New Generation engines) 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 2 (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 translation engines now integrate a routine which convert the 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 3 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 3 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 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 (see section 3.3).

2.3.3 Declarative linguistic rules

The linguistic rules in SYSTRAN’s classical approach are written directly in C, at a very low level. This makes them very powerful since they have full control over the linguistic procedures. However, the visibility is low and the learning curve is very steep.

Ideally, everything should be described in a higher level formalism, easily accessible to linguists. The linguists should not have to deal with implementation details that are not related to linguistics. This is the main concept of NG.

Given the amount of rules available in the system, and also the way they work, it is very hard to create this formalism. Covering them with a declarative approach would mean re-inventing a kind of programming language that would not necessarily be easier to work with. Also, the time needed for such a conversion is a considerable investment. Therefore, as a compromise, new rules may be written declaratively, while keeping the available rules as they are.

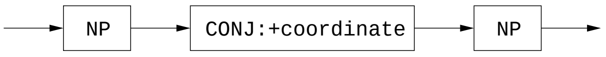
The formalism for these rules is easy to use and will eventually replace the classical approach. In the meantime, it does however give a possibility for additional development.

This is possible using the common data structure for the syntagm structure described in the previous section.

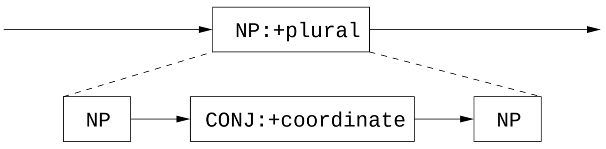
Example of a simple declarative rule:

```
<Rule id="NP11" confidence="0.9">
 <Match>
   [NP] [CONJ:+coordinate] [NP]
 </Match>
 <CreateSyntagm pos="NP">
  <Feature name="plural"/>
 </CreateSyntagm>
</Rule>
```

This rule would group the sequence



into the syntagm



The declarative rules are loaded at runtime. This means that they are easy to modify and test, and therefore provide a very quick linguistic development cycle.

The available types of parts of speech and their attributes are defined in another module, which handles the data structure for linguistic attributes and classes. This definition is also loaded at runtime.

The attribute definition module allows for a means of checking the consistency of the rules,

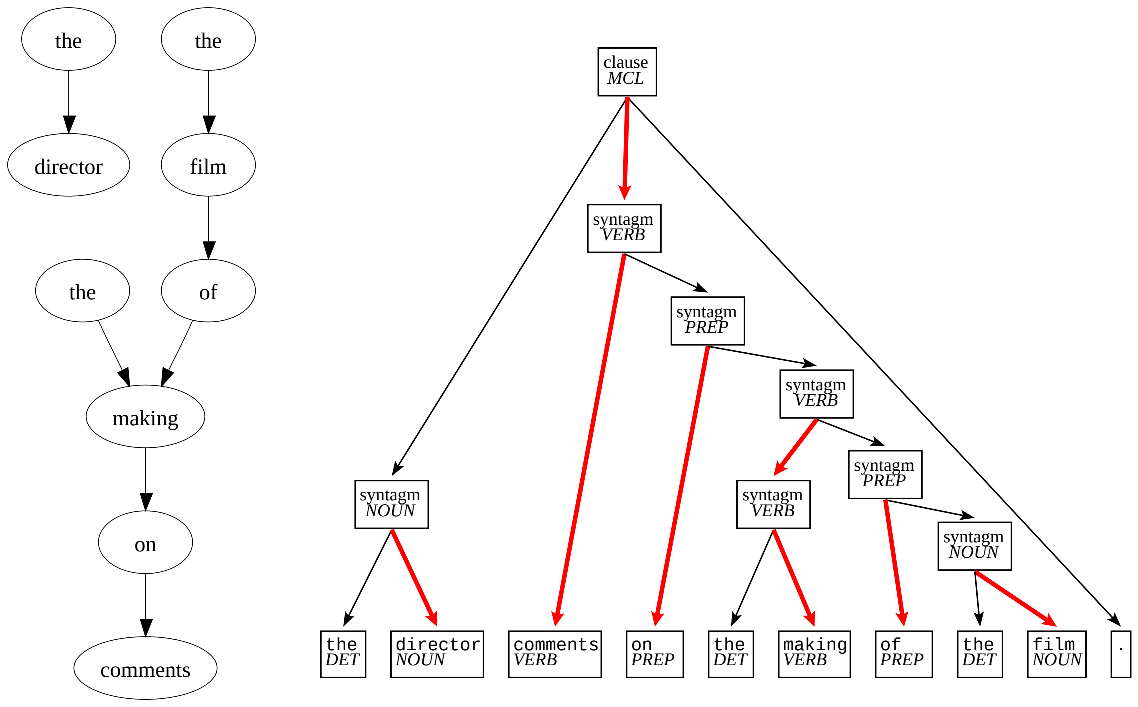


Figure 2: On the left, graph of the relations between words in the sample sentence "The director comments on the making of the film." (the relation types are not shown). On the right, syntagm tree for the same sentence, derived from the graph; head words of each syntagm are linked to the syntagm with a bold edge.

```
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 3: Algorithm for converting a syntactic relation graph to a syntagm DAG

as well as making sure that the state of the attributes is valid. For example, if the attribute *plural* is set, it should not keep the attribute *singular* if that was set by default.

Another application for this module is semantic taxonomy, where attributes can be defined to imply other attributes. For example, if there is a semantic attribute *woman*, the attributes *human* and *animate* could be defined as semantic parents.

2.4 Generic Lookup Operator

In Figure 1, the rightmost box represents a generic component performing "lookup" for the different modules.

This component called *XmlFind* is used for dictionary lookup, rule triggering, and complex morphology description. This mechanism is very generic and extremely powerful — based on finite state automaton technologies, it performs extended linguistic regular expressions. Those expressions applies on the XML structure used for the workflow and has subsequently access to most linguistic information and meta-information on the current sentence. Figure 4 describes for instance how "normalization" rules are internally represented using finite state transducers.

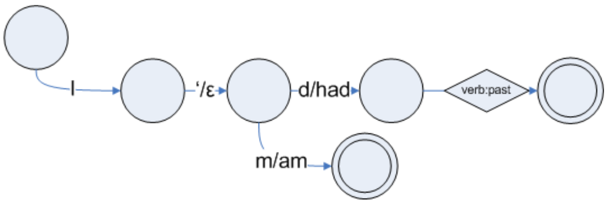


Figure 4: Finite state transducer describing normalization rules: *I'd* is normalized into *I had* when followed by a past participle. This last constraint is represented using a special transition.

Simplest transitions are letters but the following special "operators" can also be used:

- character properties operators (for instance translation memory takes into account characters properties for selecting best match)
- Unicode operators - generic Unicode regular expression (level 1)
- XPath operators on the complete sentence XML
- predefined "lookup operators" (chemical compounds, numbers, date...)
- sub-matching: possibility of calling recursively a sub-graph as a part of the current entry. This features allows for representation of local grammars as described in (Gross, 1997)
- any request on the current XML node (containing information accumulated during the process)

Any lookup performed by any module can have access to this whole set of lookup tools — this means that even "simple dictionary entries" have access to those complex operators for restricting/extending matching conditions.

In addition, the automaton can be combined on the fly with "lookup rules". These lookup rules are also described using a finite state transducer and describe the possible variants allowed during the lookup. These variants can be associated to a matching cost. Figure 5 represents such a transducer for some basic spell-checking rules.

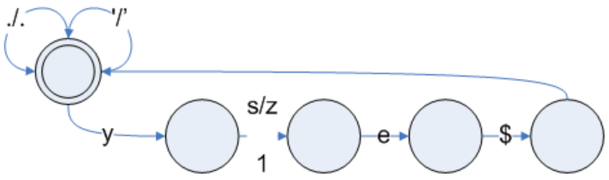


Figure 5: Simple spellcheck rules: a/ right quote and straight quotes are equivalent. b/ *yze* can be matched by *yze* with a cost 1.

3 Sample Applications

In this section, we illustrate on three simple examples, how the new architecture integrating classical translation engines allow to efficiently improve translation quality and interact with existing rule set.

3.1 Linguistic improvement by input simplification

One simple example of the immediate outcome of the architecture as described above is the linguistic improvement coming from the simplification of the input. For instance, it is possible to introduce a module in the workflow specialized in date description or other "local grammar". For dates, for instance, we have introduced modules specialized in parsing, analysis and generation of dates. Those modules interact with the complete analysis by posting a simplified representation of the recognized expression in the workflow together with syntactic and semantic information used by the native "date lexical routine".

In contrast to an external "entity recognition" approach that could substitute date by a keyword, this approach communicates information to the internal routines about the properties of the entities. Reciprocally, the linguistic code in charge of the sentence generation can communicate, in the same way, information to the date generation module — for instance case or agreement constraints.

Finally, those modules can perform a partial recognition by letting ambiguous sequences be normally processed by the existing mechanisms.

This simple example shows the simplicity with which an external mechanism can interact with existing code and customize the translation process.

3.2 Introduction of control mechanisms

A second example is the new possibility of interacting with the translation engine. For instance, in version 5, the user has the possibility of seeing homographies resolved by the analysis and interact with choice of the system. This interaction mechanism is started by the *posting* of homography information by the routine resolving the ambiguity. The XML markup carries this information which is presented to the user for validation. In the case the user wants to modify system choice, the user choice is posted in the XML workflow, and this information will be used by the homography resolution routine as a priority rule. Figure 6 shows user review and selection of homography resolution. On the same model, any system choice can be published and manually modified through user interaction.

3.3 Integrating external rules - customizing the system

Using the permanent synchronization between the classical analysis area and the high level tree representation, and by defining appropriate hooks in the linguistic workflow, it is possible to add modules that independently modify the analysis of the sentence — or add new transfer patterns — before giving the control back to the existing workflow. Hence, adding special linguistic descriptions specific to a customized domain or modifying dynamically normal system choices is fully possible.

For instance, one possible use of this extension possibility that we are currently working on, is to use statistical methods to validate and also extract new rules using corpus.

4 Conclusions and Outlook

We have described in this paper the mechanism we use to open a complex MT system and to integrate it in an open workflow. The integration process benefits from the linguistic rule set, provides "control mechanisms" (used for module communication, or interaction with the user), allows for an easy plug-in of external modules in charge of specific linguistic tasks (leading to more efficient system customization, or to combining a rule-based approach with a statistical-based approach), and reveals all intermediate results of the translation process in a structured format to the user.

To complete this architecture, the current development focal points are:

- Improving the value of existing rules by associating ponderation to each of them. Today it is possible to know the system's choices and to modify them. The next step is for us to know the system's hesitations. In a post-editing workflow, the reviewer would be able to focus on certain areas of the translation output, which would yield to higher productivity. The same ponderation can be used to calculate the global confidence level for the complete sentence translation, or can be used to decide that some decision should not be taken, and that the different possibilities be analyzed in parallel.
- Continue the modularization effort on the linguistic code in order to obtain smaller "agents". This will reduce the time required to generate new language pairs, and will enable a deeper interaction with the existing set of rules.

References

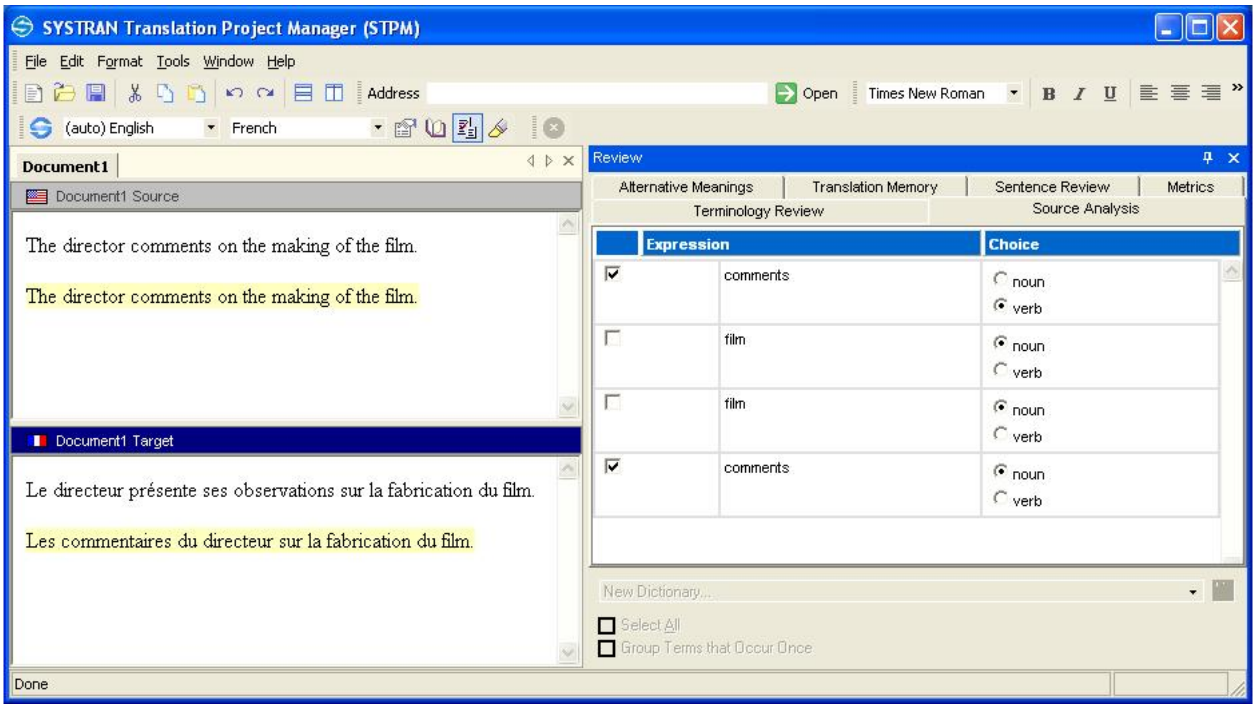
M. Gross, 1997. *Finite-State Language Processing*, chapter The Construction of Local Grammars, pages 329–354. MIT Press.

T. Váradi J. Senellart, P. Dienes. 2001. New Generation SYSTRAN translation system. In *MT Summit VIII Proceedings.*

L. Romary J. Senellart, C. Boitet. 2003. SYSTRAN new Generation: the XML translation workflow. In *MT Summit IX Proceedings.*

Jing-Shin Chang Keh-Yih Su. 1999. A customizable, self-learnable parameterized mt system: the next generation. In *MT Summit VII Proceedings.*

The Unicode Consortium. 2003. *The Unicode Standard. Version 4.0.* Addison Wesley Publishing Company.



(auto) English → French

Document1 Source

The director comments on the making of the film.

The director comments on the making of the film.

Document1 Target

Le directeur présente ses observations sur la fabrication du film.

Les commentaires du directeur sur la fabrication du film.

	Expression	Choice
✓	comments	noun [✓ verb]
	film	[✓ noun] verb
	film	[✓ noun] verb
✓	comments	[✓ noun] verb

Figure 6: User review of system choices and selection of homography resolution within SYSTRAN Translation Project Manager (version 5)