Extended Translation Memories for Multilingual Document Authoring

Jean-Luc Meunier, Marc Dymetman
Xerox Research Centre Europe
6 chemin de Maupertuis, Meylan, France
E-mail: jean-luc.meunier@xrce.xerox.com, marc.dymetman@xrce.xerox.com

Abstract

This paper discusses multilingual document authoring, viewed as providing computer support for a user to author a document in some source language while automatically generating the same content in one or many target languages. A kind of unanticipated use of multilingual authoring appeared in the service sector, in situations where an employee is servicing customers by answering their requests, or helping them, via written electronic communication. Decoupling the employee's language from the customer's language may open up new perspectives and motivated this work, where we propose a small set of extensions to be made on a translation memory to support multilingual authoring more efficiently. We describe how an instance of such extended formalism can be conveniently created thanks to a domain specific language and describe how we implemented a full system. Finally, we report on the experiment we ran in a real business setting.

Keywords: Translation Memory, Writing assistance, Multilingual Authoring, Experimentation, Domain Specific Language

1. Introduction

The need for efficiently producing a document in multiple languages most probably appeared long time ago, and the Rosetta Stone is a famous example of this need. One conventional approach to the problem consists in an authoring step followed by a translation step. With the advent of computers and computer science, new tools emerged, and authoring support tools, translation memories and machine translation are particularly relevant with this respect. A new approach emerged in the 90s, which aimed at providing computer support for authoring a document in multiple languages, merging two steps into a single activity. One early publication from Hartley and Paris (1997) says it all in its title: "Multilingual document production from support for translating to support for authoring".

The work presented here contributes to this approach by extending translation memories for use in multilingual authoring support. We will first introduce a motivating business use that was probably not imagined in the 90s, before giving some background on an existing multilingual authoring tool. We will then describe how to extend a translation memory for multilingual authoring and report on the experiment we ran in a real business setting.

2. Motivation

A kind of unanticipated use of multilingual authoring appeared in the service sector, in situations where an employee is servicing customers by answering their requests, or helping them, via written electronic communication. This situation is very common in sectors like customer care, human resource, finance, etc. The customer, or more generally requestor, contacts the agent by email, or by filling in a web form. The agent uses dedicated tools, e.g. a knowledge base or some customer relationship management tool, in order to fulfill the request and provides the requestor with a written answer. Some requests may need multiple cycles of communication, forming a conversation. So far, agents were grouped into language teams in one or several helpdesk centers and each team was sized to answer the peak load and cover for the opening hours of the customer service.

With the globalizing market, the number of serviced languages is increasing and finding agent speaking the required language(s) often becomes problematic. Since companies try to avoid opening one helpdesk per language/country they service but rather look for ways to centralize the helpdesks in one or a few helpdesk center(s), they often face the problem of finding in a certain country an agent speaking a language that is not generally spoken in that country. To accommodate with organizational issues, those agents are often also required to speak the language of the country or the company. Finding a person with the required technical and language skills can prove quite difficult and may require paying a premium to get the person onboard.

Breaking the language barrier and allowing an agent who does not speak the requestor's language to provide him/her with the required help is therefore attractive to companies operating in this business sector, even if the solution allows for handling only a portion of the total volume of requests.

Machine translation ideally should answer this need: a request could be automatically translated into the agent's language and vice-versa for the agent's answer. Practically, coping with translation errors is both critical and not easy. We distinguish two situations with different constraints: inbound and outbound correspondence.

For inbound, the request needs to be translated in the agent's language so that the agent understands the request and feels confident about his/her understanding. No need for a perfect translation quality. In usual quality evaluation terms, the fluency of the translation is of less importance than its adequacy, which can be critical.

For outbound correspondence, the translation quality that is required is much higher since the company is sending a

written answer to a customer. Both fluency and adequacy are important and the consequence of any translation errors must be carefully assessed before rolling out such a system. Although automatic confidence estimation (Blatz et al., 2004) of the translation could play a role, we have chosen a different approach based on multilingual authoring with the goal of allowing the agent to author a reply in both her/his language and in the customer's language. In term of reply's quality, the multilingual authoring tool will bring the language knowledge while the agent will bring the subject matter expertise. The goal is to create a high quality reply, both at language- and semantic-levels, so that it is not perceptible that the agent does not speak the customer's language.

In the rest of the paper, we will focus on the use of multilingual authoring for supporting the outgoing correspondence. More precisely, we focus on how to extend translation memories for setting up a multilingual authoring support system.

3. Background: the MDA Tool

Before introducing how a translation memory can be extended for supporting multilingual authoring, let us introduce here one pre-existing tool called MDA (Brun et al., 2000), which stands for Multilingual Document Authoring. This tool was conceived in the years 1998-2002. It allows a monolingual user to interactively produce a document in multiple languages, including a language s/he masters, following a document template that controls both the semantics and the realization of the document in multiple languages.

This section describes the MDA tool and its template inner working, by using excerpts of the publication "Document structure and multilingual authoring" by Brun, Dymetman and Lux (2000), so as to introduce the challenges one faces to support multilingual authoring.

In the next section, we will relate the extended translation memory formalism to this tool's template.

3.1 Approach

First, the main requirement for such a tool is that the authoring process is monolingual, but the results are multilingual. At each point of the process the author can view in his/her own language the text s/he has authored so far. This is in line with the WYSIWYM (What You See Is What You Mean) editing method described in (Power & Scott, 1998). In MDA, the areas where the text still needs refinement are highlighted and menus for selecting a refinement are also presented to the author is his/her own language. Thus, the author is always overtly working in the language s/he knows, but is implicitly building a language-independent representation of the document content.

From this representation, the system builds multilingual texts in any of several languages simultaneously. This approach characterizes our system as belonging to the paradigm of "natural language authoring" (Hartley & Paris, 1997; Power & Scott, 1998) , which is distinguished from natural language generation by the fact that the semantic input is provided interactively by a person rather than by a program accessing digital knowledge representations.

Second, the system maintains strong control both over the semantics and the realizations of the document. At the semantic level, dependencies between different parts of the representation of the document content can be imposed: for instance the choice of a certain chemical at a certain point in a maintenance manual may lead to an obligatory warning at another point in the manual. At the realization level, which is not directly manipulated by the author, the system can impose terminological choices (e.g. company-specific nomenclature for a given concept) or stylistic choices (such as choosing between using the infinitive or the imperative mode in French to express an instruction to an operator).

Finally, the semantic representation underlying the authoring process is strongly document-centric and geared towards directly expressing the choices which uniquely characterize a given document in an homogeneous class of documents belonging to the same domain. The screenshot in figure 1 shows the MDA tool, with a document being authored.

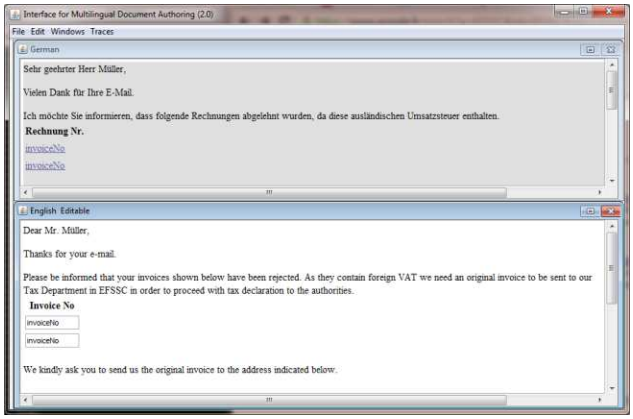


Figure 1: Screenshot of the MDA tool in use

3.2 Interaction Grammars (IG)

Let us now give some details about the formalism of Interaction Grammars used by MDA. We start by explaining the notion of choice tree on the basis of a simple context-free grammar.

3.2.1. Context-free grammars and choice trees

Let's consider the following Context-Free Grammar (CFG), ignoring for now the first column (italic text):

```
warnSymp warning --> "in case of", symptom, ",",
   action.
weak       symptom --> "weakness".
conv       symptom --> "convulsions".
hea        symptom --> "headache".
rest       action --> "get some rest".
consult    action --> "call your doctor
   immediately".
```

What does it mean to author a "document" with such a CFG? It means that the author is iteratively presented with partial derivation trees relative to the grammar (partial in

the sense that leaves can be terminals or non-terminals) and at each given authoring step both selects a certain nonterminal to "refine", and also a given rule to extend this non-terminal one step further; this action is repeated until the derivation tree is complete.

If one conventionally uses the identifier in italic in first column to name each rule, then the collection of choices made by the author during a session can be represented by a choice tree labelled with rule identifiers, also called combinators. An example of such a tree can be written `warnSymp(weak, rest)` reflecting the generation of the text "`in case of weakness, get some rest`".

3.2.2. Making choice trees explicit

Choices trees are in MDA the central repository of document content and we want to manipulate them explicitly. Definite Clause Grammars (DCG) (Pereira & Warren, 1980) represent possibly the simplest extension of context-free grammars permitting such manipulation.

Our context-free grammar can be extended straightforwardly into the DCG[1].

```
warning(warnSymp(S, A)) --> "in case of",
   symptom(S), ",", action(A).
symptom(weak) --> "weakness".
symptom(conv) --> "convulsions".
symptom(hea) --> "headache".
action(rest) --> "get some rest".
action(consult) --> "call your doctor
   immediately".
```

What these rules do is simply to construct choice trees recursively. Thus, the first rule says that if the author has chosen a symptom through the choice tree S and an action through the choice tree A, then the choice tree `warnSymp(S, A)` represents the description of a `warning`.

If now, in this DCG, we "forget" all the terminals, which are language-specific, by replacing them with the empty string, we obtain the following "abstract grammar":

```
warning(warnSymp(S, A)) --> symptom(S),
   action(A).
symptom(weak) --> [].
symptom(conv) --> [].
symptom(hea) --> [].
action(rest) --> [].
action(consult) --> [].
```

This grammar is in fact equivalent to the definite clause program:

```
warning(warnSymp(S, A)) :- symptom(S),
   action(A).
symptom(Weak) .
symptom(conv) .
symptom(hea) .
action(rest) .
action(consult) .
```

This abstract grammar (or, equivalently, this logic program), is language independent and recursively defines a set of well-formed choice trees of different categories, or types. Thus, the tree `warnSymp(weak, rest)` is well-formed "in" the type `warning`.

3.2.3. Dependent Types

In order to stress the type-related aspects of the previous tree specifications, we are actually using in our current implementation the following notation for the previous abstract grammar:

```
warnSymp(S, A)::warning --> S::symptom,
   A::action.
weak::symptom --> [].
conv::symptom --> [].
hea::symptom --> [].
rest::action --> [].
consult::action --> [].
```

The first rule is then read: "if S is a tree of type symptom, and A a tree of type action, then warnSymp(S, A) is a tree of type warning", and similarly for the remaining rules.

The grammars we have given so far are deficient in one important respect: there is no dependency between the symptom and the action in the same warning, so that the tree is `warnSymp(weak, rest)` is well-formed in the type address. In order to remedy this problem, dependent types (Ranta, 2004) can be used. From our point of view, a dependent type is simply a type that can be parameterized by objects of other types. We write:

```
warnSymp(S, A)::warning -->
   S::symptom(Severity), A::action(Severity).
weak::symptom(mild) --> [].
conv::symptom(severe) --> [].
hea::symptom(severe) --> [].
rest::action(mild) --> [].
consult::action(severe) --> [].
```

We have introduced a `Severity` parameter that is shared by the two type symptom and action forcing certain associations between a given symptom and a given action.

3.2.4. Parallel Grammars and Semantics-driven Compositionality for Text Realization

We have just explained how abstract grammars can be used for specifying well-formed typed trees representing the content of a document.

In order to produce actual multilingual documents from such specifications, a simple approach is to allow for parallel realization English, French, … grammars, which all have the same underlying abstract grammar (program), but which introduce terminals specific to the language at hand. Thus the following French and English grammars are parallel to the previous abstract grammar[2]:

```
warnSymp(S, A)::warning --> "In case of",
   S::symptom(Severity), ", " ,
   A::action(Severity) , "." .
weak::symptom(mild) --> "weakness".
conv::symptom(severe) --> "convulsions".
hea::symptom(severe) --> "headache".
rest::action(mild) --> "get some rest".
```

[1] According to the usual logic programming conventions, lowercase letters denote predicates and functors, whereas up-percase letters denote metavariables that can be instanciated with terms

[2] Because the order of goals in the right-hand side of an abstract grammar rule is irrelevant, the goals on the right-hand sides of rule in two parallel realization grammars can appear in a different order, which permits certain reorganizations of the linguistic material (situation not shown in the example).

```
consult::action(severe) --> "call your doctor".
```

```
warnSymp(S, A)::warning --> "En cas de",
   S::symptom(Severity), ", " ,
   A::action(Severity) , "." .
weak::symptom(mild) --> "fatigue".
conv::symptom(severe) --> "convulsions".
hea::symptom(severe) --> "maux de tête".
rest::action(mild) --> "prenez du repos".
consult::action(severe) --> "consultez votre
   médecin".
```

The logic programming representation of such a grammar has rules of the following form:

```
a1(B,C,...)::a(D,...)-english[X,Y, ...] -->
          B::b(E,...)-english[X, ...] ,
          ". . ." ,
          C::c(F,...)-english[Y, ...] ,
          ...
          {constraints(B,C,...,D,E,F,...)},
          {conditional_code(X, Y, ...)}.
```

Those rules are close to the grammar rules, with additional language-specific parameters to deal with constraints that are specific to one language.

As the reader can see, the creation of a MDA template was a complex task, requiring unusual skills, namely the knowledge of definite clause grammars and Prolog. On the other hand we were attracted by the power of the tool and chose to use it as target platform for our new formalism.

4. Extending Translation Memories

While the interaction grammars (IG) presented above proved to apply well to the problem of modelling agents' replies, or more generally agents' language, their creation was somehow complex and requiring uncommon expertise. We therefore looked for some alternative formalism. In particular, we considered the structure of a translation memory, since it intrinsically captures the desired parallelism between one source language and some target(s) one(s). It however lacks of the power of a grammar to define or guide the agent's language. We have therefore defined a minimal set of mechanisms that should be added to a translation memory structure to support our goal.

The proposal consists in following a Translation Memory (TM) paradigm, with a set of extensions towards supporting the creation of document template for multilingual document authoring by a monolingual user. Our aim is to facilitate the design of document grammar for multilingual document authoring by non-experts

More precisely, where a translation memory stores document fragments together with the corresponding translation, our extension consists in adding the notion of fragment type, allowing a fragment to be generalized to a certain type of textual content; we also introduce the notion of global variable, allowing some textual contents to be shared across a document. Each fragment remains aligned with its counterpart(s) in the other language(s). Additional mechanisms include constraints and conditional realization.

Without loss of generality, let's consider the case of generating some document in English and French.

We will call 'designer' the person in charge of designing a document grammar, which can then be used by a 'user' of the MDA tool

4.1. A translation memory approach with Context Free Grammar power

Where a standard translation memory would be a two-columns table, with parallel segments in English and French, our extended TM will be a sequence of four-columns tables:

- Column 1 is the so-called `case`: it uniquely identifies, and labels, a specific row within a table.
- Column 2 is the so-called `wizard`: it is used to guide the interaction between the multilingual authoring tool, e.g. legacy MDA, tool and the user, when she/he authors a new document.
- Column 3 and 4 are the `English` and `French` columns: they contain the realizations (concrete realizations as character string) of the segment in the two languages.
- Each additional language would require one addition column.

Each such table is called a `type` and has a unique name as well. See the table named "MyType" in figure 2. Some common types such as `STRING`, `NUMBER` and `DATE` are pre-defined in the formalism and in the tool.

The underlying formalism has ties with Context Free Grammars (CFG), since a type can be seen as a CFG non-terminal, while the cases correspond to enumerating and naming the possible production rules for that non-terminal. More precisely, this formalism has ties with Synchronous Context Free Grammar (Chiang & Knight, 2006).

Let's consider a simple CFG grammar like:

```
Document -> Det Noun Adj "."
Det -> "one"
Det -> "two"
Noun -> …
…
```

We would express such a CFG as the sequence of tables shown in figure 3.

We see that the wizard allows the template designer to associate a question with a given type. Typically, in the MDA tool (when a user authors a new document), the tool will display the question and propose (some or all of) the case names for that type as possible answers to the user.

The English and French columns of a case can refer (zero or multiple times) to the types listed in the `wizard` part of the case, in any order, and can interleave them with terminal strings. In the previous example, observe how the English and French realizations re-order the non-terminals.

We will call 'type call' a non-terminal in the Wizard, English and French columns, since it can be seen as

'calling' a type that is defined in its own extended TM table.
In addition, because the English and French refer to the wizard type calls, it may be necessary to distinguish multiple calls to the same type, e.g. for a rule like `Document -> Det Noun Verb  Det Noun`.
So a type call may be named for further reference within the same case from the English or French realization, as for instance in figure 4.
This Translation Memory Grammar (TMG) approach makes one step towards supporting multilingual document authoring using parallel context-free grammars, but requires additional mechanism to be available, as we will see below.

4.2. A translation memory approach with Interaction Grammar power

We are here extending our TMG formalism to support dependencies between types as well as dealing with extra conditions on the realization in natural language. As explained in section 3.2, the existing MDA tool relies on the so-called Interactive Grammars (IG) formalism, which is a specialization of the Definite Clause Grammars (Pereira & Warren, 1980) inspired by the GF formalism (Ranta, 2004). Please refer to (Brun et al., 2000) for full details on this formalism.
We reproduce below the IG abstract grammar (which does not shows terminals) of the drug warning example:

```
warnSymp(S,A)::warning -->
   S::symptom(SympClass),
   A::action(SympClass).
weak::symptom(mild)    --> [].
conv::symptom(severe) --> [].
hea::symptom(severe)  --> [].
rest::action(mild)     --> [].
consult::action(severe) --> [].
```

We propose here a simple way to inject some key aspect f the IG formalism in our TM-based formalism to deal with dependencies among types.
For doing so, a type may have one or multiple attribute(s), the value of which can be constrained by an equality operator. The constraint can involve an attribute and a constant or two attributes. Note that the '`=`' operator is asserting a constraint rather than expressing an assignment.
So the above example would be reflected as shown in figure 5.
Scoping: the attributes of a type are accessible from the type itself using the keyword `this`, or via a reference of a wizard's type call within a case. An attribute set in the wizard column is visible in other columns, while if set in the 'French' column, it will only be visible from a 'French' column.
Moreover, it is common when designing a grammar to require access to certain information from several different places. Typically, when designing a template of a letter to a customer, the designer may need to access the customer name from several parts of the documents, which will typically correspond to accessing it from several types of the TM-like template.
We therefore introduce one more mechanism allowing the designer to declare a so-called `global` by associating a (grammar-)unique name with a type. This name can then be used as reference in any case of any type.
Back to the drug warning, the designer could have for instance declared `DrugName` as a global of type `STRING` to conveniently insert the name of the drug in a realization.
In addition, the designer could have declared a global `DrugForm` of type `pharm_form` (see in next section) to reflect the pharmaceutical form of the drug (tablet, capsule, syrup, eye drop).

4.3. Conditional Realization

We introduce the last mechanism to deal with fine realization issues. Typically, in French the noun 'tablet' has a genre which must be taken into account by a related adjective or past-participle (among others…).
We introduce conditional realization, where the designer can condition the realization by constraints on attributes. (The constraint is enforced locally to the case, unless it involves a global.)
The example in figure 6 below illustrates this.
The generated grammar also includes a catch-all mechanism so that if no condition is met, some error message is produced and shown to the user.
With such formalism, the interaction grammar example given in section 3.2 is shown in figure 7.

We believed this formalism to considerably alleviate the complexity of defining the resource required to support multilingual authoring and were interested in testing this belief, as described in next sections.

5. Implementation: dedicated tool suite for the TM Grammar

Editing such a TM grammar is not straightforward because of its structure as well as the multiple inner references to types, attributes, etc. We therefore decided to create some dedicated editing tool.

5.1. XML Lingua

First an XML representation was defined thanks to a RelaxNG (Clark & Murata, 2001) XML schema. Any TMG (translation-memory grammar) expressed in this XML language can then be displayed in the above tabular structure thanks to a CSS stylesheet.
We then explored the possible use of some off-the-shelf schema-aware XML editor, but none were supporting the CSS view in editing mode. So the use of an XML representation was both convenient and good engineering practice but was not appropriate for editing purpose.

5.2. Domain Specific Language

We therefore decided to design a Domain Specific Language (DSL) for our translation-memory grammars and implemented it using the Eclipse/Xtext/Xtend framework (www.eclipse.org/org). Eclipse is an "*an open*

development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle". Xtext is "*a framework for development of programming languages and domain specific languages*". Xtend is "*a flexible and expressive dialect of Java*".

The result is an editor with syntax coloring, content assistance, outline, validation and quick fix facilities integrated into the Eclipse IDE, which comes with rich functionalities for versioning etc, and able to generate the XML representation of a translation-memory grammar.

In Xtext, designing a DSL involves specifying a particular kind of BNF for the language to describe the concrete syntax and how it is mapped to an in-memory representation - the semantic model. This model will be produced by the parser on-the-fly when it consumes an input file. The full-fledged editor and required parser are automatically generated from the special BNF.

In Xtend, one can further enrich the editor, for instance to define the outline view appearing on the right panel in the screenshot below. But more importantly, we used Xtend to automatically generate the XML corresponding to a TMG being edited.

In order to generate the IG grammar required for the MDA tool given a TMG instance, we specifically developed a compiler from XML to IG.

Figure 8 shows the same Symptom/Action example created within this DSL.

6. Experiment

We experimented the MDA tool and the translation-memory grammar (TMG) with the help of colleagues from Xerox service who are running the Account-Payable office of a Xerox customer. In this office, Xerox agents are receiving emails from suppliers of the Xerox customer regarding invoices, payments, etc. The agents use the customer database and IT infrastructure to answer the requestors by email as well. The contractual language is German and this was requiring the agents to be fluent in German in addition to the job-specific skills.

Xerox service was interested in testing if combining machine translation and multilingual authoring would allow a monolingual English-speaking agent to work in this context where the business language, contracted by the customer, is German. More precisely, the goal was to evaluate the proportion of replies that could be handled by an English agent using MDA, assuming the machine translation of the request was satisfactory. Should the translation be unsatisfactory or MDA inappropriate to author a reply, then the request would be escalated to a German-speaking agent.

With the aim of handling the highest possible proportion of replies, the service team provided us with a typology of replies and selected the most frequent types for us to encode those types in a TMG. Given this list of pairs of (English, German) texts, we then devised a TMG. Looking at the regularities, we structured each reply as a sequence made of: greetings, thanks?, message+, ending (where ? denotes an optional item and + an item occurring one or more times). We identified 5 different forms of greetings and ending. The core of the reply could be structured further into 6 sub-types, totalizing 90 cases, as they are called in TMG.

In order to jointly design the TMG with the Xerox service team, we exposed them to the TMG thanks to the tabular view created by use of the CSS on the XML file. Despite some of our colleagues were not IT expert, the tabular structure was easy to understand. So we ended up exchanging annotated document, namely MS-Word document in track change mode, so as to work jointly on the TMG. We show in figure 9 an excerpt of such a document sent back from our service colleagues who fixed the German side of the case "AP13_Missing_Invoice".

Three rounds of tests were required to reach a satisfactory level, after a dozen of exchange of the TMG between the research and service teams. For each tests, the service team evaluated if a reply was both doable with MDA and of acceptable quality, on about 150 requests, by asking a monolingual English agent to answer a (machine-)translated request.

The table below summarizes the results:

Test results	**Round 1**	**Round 2**	**Round 3**
Outbound unacceptable	81%	42%	7%
Outbound acceptable	19%	58%	93%

The creation of the first version of TMG took about 4 days of work, while the following two next versions took 2 days each. The result obtained at round 3 is quite satisfactory. The use of a human-readable tabular structure proved to be valuable in this context where actors with different expertise, linguistic/business/IT, need to cooperate.

However, the TMG we created remains rather simple in the sense that only few semantic constraints and linguistic difficulties were to be handled. Actually, this relatively low complexity may also be characteristic of the domain of application because agents' discourse often follows some company policy.

It remains unclear how well the TMG can scale to more sophisticated and advanced answer writing since the complexity of the grammar may become too high for handcrafting it. In 2000 Brun et al. chose a rather complex example involving pharmaceutical notices. We believe this example would be much easier to write with the TMG than with the 2000 original formalism. We are looking forward to new example of practical use to answer this important question.

7. Conclusion

In this paper we have presented a novel formalism for multilingual authoring so as to support a user in creating a document in "his" language while automatically generating the same content in some foreign language(s). The proposed formalism consists in a translation memory structure with a minimal set of additional mechanisms, to form what we call a Translation Memory Grammar (TMG).

To operationally implement it, we have relied on a pre-existing tool called MDA and on its underlying interactive grammars (IG), themselves implemented in a logic programming language. While logic programming was convenient, we believe there are alternative ways to

implement our proposed formalism.

To support the editing of the TMG, we have devised a domain specific language using modern software engineering techniques.

Since we introduced this tool in the context of a particular business need, we have described the experiment we did with our colleagues from the service arm of our company, in the context of a contracted provision of service to an external customer.

From the experiment, we draw the following conclusions:

- The tabular structure is valuable for supporting the necessary interaction between team members with different and complementary expertise: linguistic (source and target languages), business (Account payable here), IT (for creating the TMG).
- Basic linguistic phenomenon can be captured by simple syntaxic encoding in the tabular structure, provided the IT person has rudimentary knowledge of both the source and target languages.
- The Eclipse/Xtext/Xtend framework allowed us to create a robust DSL.
- The Translation Memory grammar was powerful and expressive enough for answering these business needs.

Unfortunately, at the time of writing of this article we have no feedback from the field regarding the user acceptance of the tool and how the new practice compares to previous one in term of effort/resource. On the other hand, during test phases, no concern was raised regarding this matter, so we are optimistic.

We are now looking forward to experimenting with transferring the TMG editing tool suite to our service colleagues so as to validate the use of this formalism by non-specialists.

8. Acknowledgements

We thank Caroline Brun and Veronika Lux as their work and publications on MDA, jointly with Marc Dymetman, are central to the present work. We are also thankful to our colleagues from Xerox service for their participation in the experiment.

9. References

Blatz, J., Fitzgerald, E., Foster, G., Gandrabur, S., Goutte, C., Kulesza, A., Sanchis, N., Ueffing, N. (2004, August). Confidence estimation for machine translation. In Proceedings of the 20th international conference on Computational Linguistics(p. 315). Association for Computational Linguistics.

Brun, C., Dymetman, M., & Lux, V. (2000, June). Document structure and multilingual authoring. In Proceedings of the first international conference on Natural language generation-Volume 14 (pp. 24-31). Association for Computational Linguistics.

Chiang, D., & Knight, K. (2006). An introduction to synchronous grammars. Tutorial available at http://www. isi. edu/□ chiang/papers/synchtut. pdf.

Clark, J., & Murata, M. RELAX NG Specification. Oasis, December 2001.

Hartley, A., & Paris, C. (1997). Multilingual document production from support for translating to support for authoring. Machine Translation, 12(1-2), 109-129.

Pereira, F. C., & Warren, D. H. (1980). Definite clause grammars for language analysis—a survey of the formalism and a comparison with augmented transition networks. Artificial intelligence, 13(3), 231-278.

Power, R., & Scott, D. (1998, August). Multilingual authoring using feedback texts. In Proceedings of the 17th international conference on Computational linguistics-Volume 2 (pp. 1053-1059). Association for Computational Linguistics.

Ranta, A. (2004). Grammatical framework. Journal of Functional Programming, 14(2), 145-189.

10. Figures

MyType	(Wizard)	(English)	(French)
Case1-name	…	…	…
Case2-name	…	…	…
…			

Figure 2: a type named “MyType” in tabular view.

Document	(Wizard)	(English)	(French)
One-noun-phrase-document	“Choose a determiner:” **Det** “Choose a noun:” **Noun** “Choose an adjective:” **Adj**	**Det Adj Noun** “.”	**Det Noun Adj** “.”

Det	(Wizard)	(English)	(French)
Case_one		“one”	“un”
Case_two		“two”	“deux”

…

Figure 3: Example of CFG in the proposed formalism.

Document	(Wizard)	(English)	(French)
One-simple-sentence-document	“Choose a determiner:” **Det:d1** “Choose a noun:” **Noun:n1** “Choose a verb:” **Verb** “Choose a determiner:” **Det:d2** “Choose a noun:” **Noun:n2**	**d1 n1 Verb d2 n2** “.”	**d1 n1 Verb d2 n2** “.”

Figure 4: Reference to type calls

warning	(Wizard)	(English)	(French)
warnSymp	“Choose a symptom:” **symptom:S** “Choose an action:” **action:A** **S**.severity =**A**.severity	”In case of” **S** ”,” **A** “.”	"En cas de" **S** "," **A** "."

symptom	(Wizard)	(English)	(French)
weak	**this**.severity=mild	“weakness”	**…**
conv	**this**.severity=severe	“convulsions”	**…**
hea	**this**.severity=severe	“headache”	**…**

action	(Wizard)	(English)	(French)
rest	**this**.severity=mild	”take some rest”	**…**
consult	**this**.severity=severe	“consult immediately”	**…**

Figure 5: An example of constraint

pharm_form	(Wizard)	(English)	(French)
tablet		“tablet”	”comprimé” **this**.gender=m
capsule		“capsule”	”gélule" **this**.gender=f

use	(Wizard)	(English)	(French)
swallow	"select a form:" **pharm_form**:F	"Swallow the" **F** "without crunching."	"Avaler" (F.gender=f "la" | F.gender=m "le") **F** "sans croquer."

Figure 6: Conditional Realization

Warning	(Wizard)	(English)	(French)
simple	"Choose a symptom:" Symptom:S "Choose an action:" Action:A S.severity=A.severity	"In case of" S "," A "."	"En cas de" S "," A "."

Symptom	(Wizard)	(English)	(French)
weak	this.severity=mild	"weakness"	"faiblesse"
conv	this.severity=severe	"convulsions"	"convulsions"
hea	this.severity=severe	"headache"	"maux de tête"

Action	(Wizard)	(English)	(French)
rest	this.severity=mild	"take some rest"	"prendre du repos"
consult	this.severity=severe	"consult immediately"	"consulter immédiatement"

Figure 7: Example 3.2.3 fully implemented

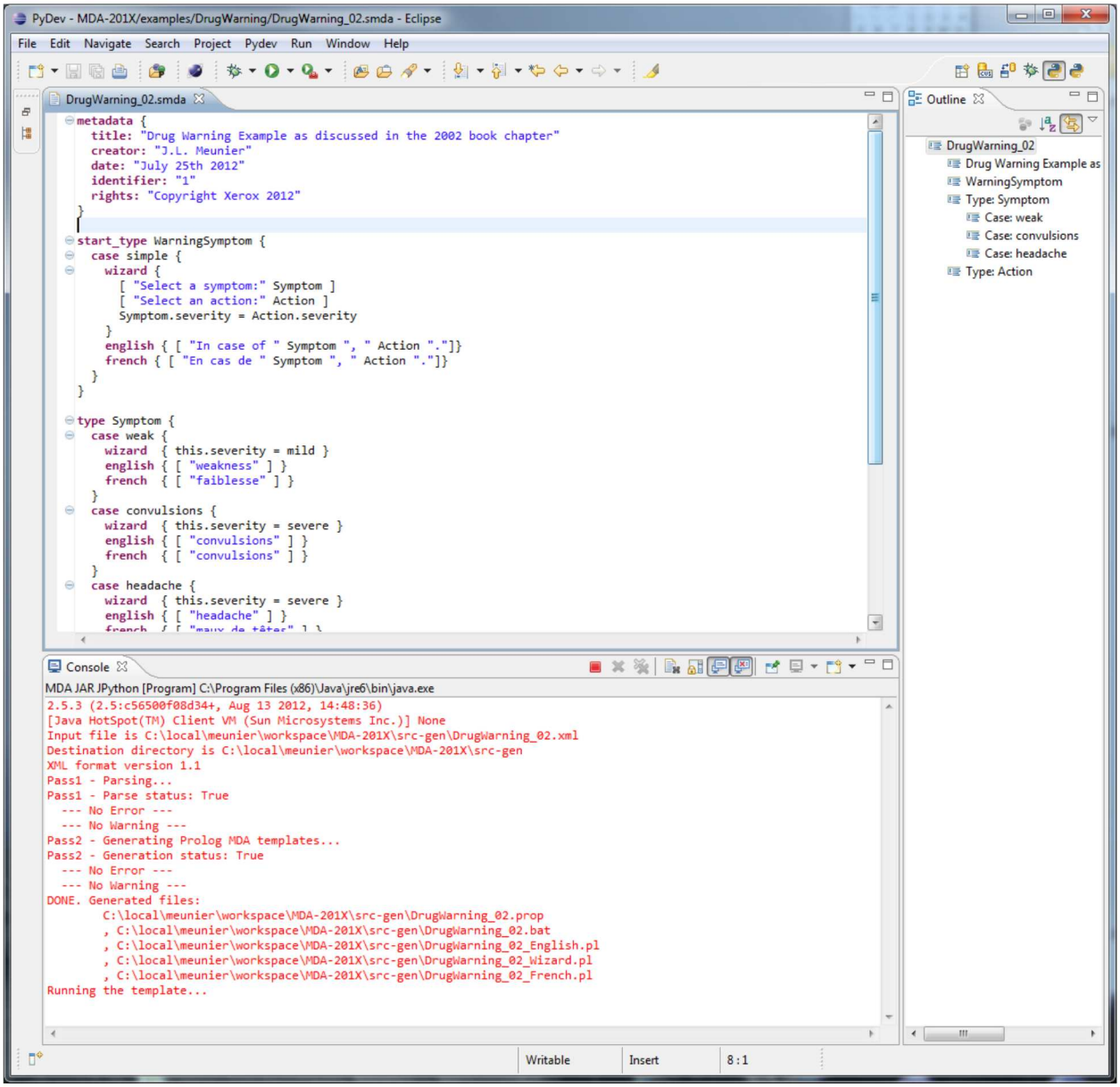
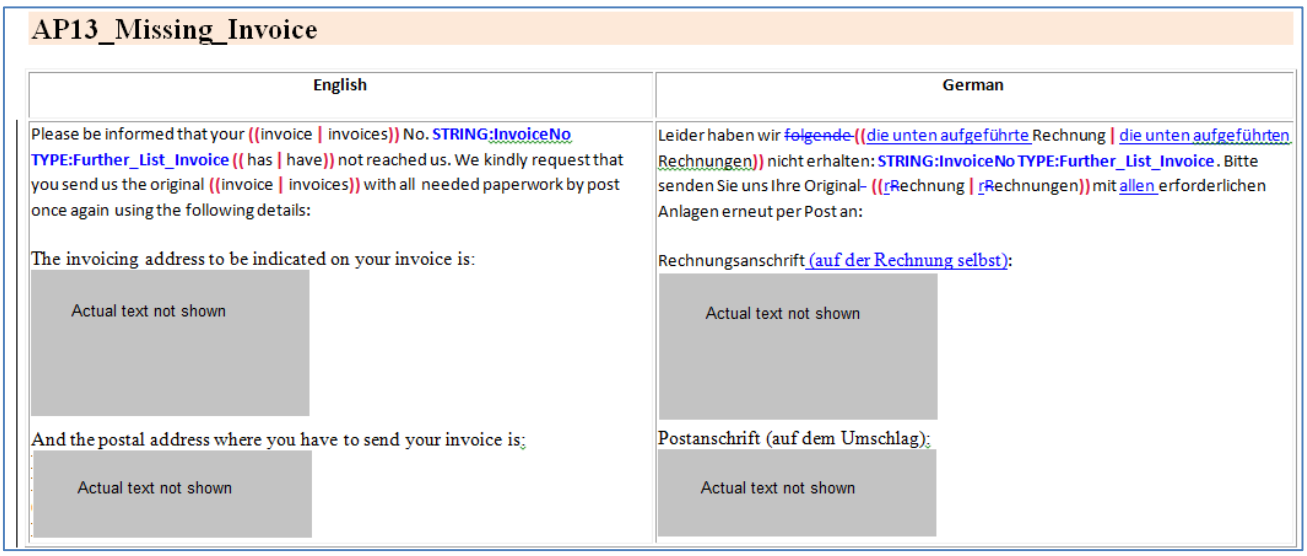


Figure 8: The DSL editor for translation-memory grammar in use, with a trace of the compiler producing the

corresponding MDA IG grammar



AP13_Missing_Invoice

English	German
Please be informed that your ((invoice | invoices)) No. **STRING:InvoiceNo TYPE:Further_List_Invoice** ((has | have)) not reached us. We kindly request that you send us the original ((invoice | invoices)) with all needed paperwork by post once again using the following details: The invoicing address to be indicated on your invoice is: Actual text not shown And the postal address where you have to send your invoice is: Actual text not shown	Leider haben wir ~~folgende~~ ((die unten aufgeführte Rechnung | die unten aufgeführten Rechnungen)) nicht erhalten: **STRING:InvoiceNo TYPE:Further_List_Invoice**. Bitte senden Sie uns Ihre Original~~-~~ ((r~~R~~echnung | r~~R~~echnungen)) mit allen erforderlichen Anlagen erneut per Post an: Rechnungsanschrift (auf der Rechnung selbst): Actual text not shown Postanschrift (auf dem Umschlag): Actual text not shown

Figure 9: MS-Word was used in track-change mode to interact with the service team. Note that conditional text, surrounded by double- red parentheses, was not an issue for them.