Multilingual Generation and Summarization of Job Adverts: the TREE Project

Harold Somers, Bill Black
Centre for Computational Linguistics,
UMIST,
Manchester, England

Joakim Nivre, Torbjörn Lager
SSKKII,
University of Göteborg,
Sweden

Annarosa Multari, Luca Gilardoni
Quinary SpA,
Milano, Italy

Jeremy Ellman, Alex Rogers
MARI Computer Systems Ltd,
Ashington, Northumberland, England

Abstract

A multilingual Internet-based employment advertisement system is described. Job ads are submitted as e-mail texts, analysed by an example-based pattern matcher and stored in language-independent schemas in an object-oriented database. Users can search the database in their own language and get customized summaries of the job ads. The query engine uses symbolic case-based reasoning techniques, while the generation module integrates canned text, templates, and grammar rules to produce texts and hypertexts in a simple way.

1 Introduction

Free movement of labour across national boundaries is an important aim of the European Union.[1] One of the prerequisites for this open labour market is accessibility of information about employment opportunities, both from the point of view of people seeking work, and of their potential employers. However, many EU citizens are denied full access to employment opportunities because information may not be readily available, and even where it is, it may not be available in the right language. The TREE project aims to address this problem by providing a system on the Internet where employers can deposit job ads, and which users can browse, each in their own language. Access to this service will be either through the user's own Internet provider, or at dedicated terminals located in employment centres. There are currently very many Internet sites where jobs are advertised, and indeed using information retrieval techniques next to natural language processing to search job offer databases is not a new application, cf. (Vega, 1990; Caldwell & Korelsky, 1994). But no other application – as far as we can discover – offers the opportunity of searching and of getting summaries of job ads in languages other than that of the original announcement.

TREE therefore offers two significant services: intelligent search and summarization on the one hand, and these independent of the original language of the job ad on the other. It could be argued that the latter at least could be achieved by hooking a commercial Machine Translation (MT) system up to an Internet employment service. Although MT has had some success on the Internet (Flanagan, 1996), this is with largely sympathetic users who understand well the limitations of MT. Its use for a more delicate task aimed at the general public, especially a public which is not necessarily highly educated, is certainly out of the question, for well known reasons which we need not explore here. Suffice to say that an experiment in Canada using an MT system for precisely this application (Murray, 1989) was far from successful.

It is also apparent that for many jobs in a location where a different language is spoken, sufficient linguistic knowledge at least to read an ad for a job in that region would be one of the prerequisites of the job: this is certainly the case for the kind of professional positions often advertised on the Internet. Nevertheless, our system offers users the possibility of searching in their own language for jobs advertised in a variety of languages. Also, there is a significant workforce for which foreign-language skills are not a prerequisite for working abroad, and which, furthermore, has traditionally been one of the most mobile: seasonal semi- and unskilled workers. For this reason, the domain we have chosen for the prototype development of the TREE project is the hotel and catering industry.

2 Overall design

The TREE system stores job ads in a partly language-independent schematic form, and is accessed by job-seeking users who can specify a number of parameters which are used to search the job database, and who can also customize the way the information retrieved is presented to them. A second type of user is the potential employer who provides job announcements to the system in the form of free text via an e-mail feed or, it is planned, via a form-filling interface (though we shall not discuss this latter input mode here).

The initial prototype system currently implemented can store and retrieve job ads in three languages – English, Flemish and French – regardless of which of these three languages the job was originally drafted in.

The system has four key components which are the subject of this paper. Telematics, HCI and certain other issues such as maintenance of the system (deleting old ads, user training, legality of texts in different countries) and the information retrieval aspects of the system will not be discussed in this paper.

The four components which we discuss here are: (a) the schema data structure for storing the job ads, and the associated terminological and lexical databases; (b) the analysis module for converting job ads received into their schematic form; (c) the query interface to allow users to specify the range of job ads they wish to retrieve; and (d) the generator, which creates a customised selective summary of the job ads retrieved in HTML format. To a great extent, the design of each of these modules is not especially innovative. However, the integration of all these functions is, from a methodological point of view, a good example of how a variety of techniques can be combined into a real application with a real use in the real world.

3 Data Structures

3.1 Job ad representation schema

Job ads are stored in the system in a "schema", which is a typed feature structure consisting of named slots and fillers. The slots, some of which have a simple internal structure of their own, identify elements of the job ad. Many, though not all of the slots can be specified as part of the search, and all of them can be generated as part of the job summary.

The fillers for the slots may be coded language-independent references to the terminological database, source-language strings which can nevertheless be translated on demand with reference to the "lexicon", or literal strings which will not be translated at all. The stylised partial example of a filled schema in Figure 1 gives an impression of

the data structure. The distinction between terms and items in the lexicon is discussed below, but we consider first the design and implementation of the schema database.

Figure 1: A partial example of a filled job schema. Slot names are shown in CAPITALS, fillers in quote marks are stored as strings; other fillers are coded.

```
JOB: waiter          JOBCODE: 92563
NUMBER_OF_JOBS: several
LOCATION: "Urmston"         WORKTIME: 2
SKILLS:EXPERIENCE:essential
APPLICATION:PHONE: 224 8619
CONTACT NAME: "Andrea"
ORIGINAL_TEXT: "Urgent!!! P/T Waiters
required, Urmston area. Experience
essential. Phone Andrea on 224 8619."
```

The main aim of the schema is to represent in a consistent way the information which the analysis module extracts from the job ads, which the query module searches, and from which the generation module produces text. Note that the example shown in Figure 1 is rather simplified for the purposes of illustration. The schema module provides a database of job schema instances (Onyshkevych, 1990). The analysis and design phases were conducted using the OMT (Rumbaugh, 1990) object-oriented methodology. Since the system currently treats three languages (with the prospect of extension to more), we decided to codify in a language-neutral fashion the information extracted from the ads, converting equivalent linguistic terms into codes and vice versa via the analysis and generation modules described below.

3.2 Terminology

The terminology module has been designed with the general aim of supporting all the common functionalities shared by the analysis, generation and query modules and of supporting a language-independent term bank to permit multilingual handling of the schema database contents. We have focused on domain-specific terms and classifications, not covering generic language issues nor providing a general lexicon and thesaurus.

Different kinds of domain-specific information can be found as slot fillers, depending on the intended meaning of schema slots. The most relevant information is obviously job types. Existing job classifications have been established for example by the European Commission's Employment Service (EURES, 1989), by the ILO (ILO, 1990) and several individual companies; each provides a hierarchical classification of jobs, specifying, for each term, a distinct code, a description of the job, one or more generic

terms commonly used to refer to the specific job, and possibly a set of synonyms. The description of the job ranges, depending on the classification, from a quite broad one to greatly detailed ones, sometimes highlighting differences existing in different countries (e.g. according to the EURES classification, a "waiter" in some EU states is also required to act as a barman while in others is not). Job classifications therefore provide at least three different kinds of information:

- Definition of recognized job types, with a (more or less) precise definition of what the job is; *chef* is a recognized item, as well as *pizza chef*, while *chef specializing in preparing hors d'oeuvres* is not; classifications are obviously arbitrary as long the boundary between whether a specific job is a recognized one or simply an "unrecognized" classification simply depends on the level of granularity the classifier decides to use.

- Classification of job types along ISA hierarchies (e.g. a *wine waiter* ISA type of *waiter*).

- Linguistic information about commonly used terms and synonyms used in a given language (or more than one) to refer to the specific term.

Accordingly, job classification terms are classified, coded (i.e. a distinct code identifying the term is associated with each term) and a list of standard "names" as well as recognized synonyms is associated with them. The classification and coding schema of VDAB, one of the end-user partners in the project, is used, but extensions deriving from other schema could obviously be envisaged. Translation tables are provided for each term, containing the names used in the different languages. Alignments across different languages are kept whenever possible. Problems due to missing equivalent terms in different languages, or to slightly different meanings, are handled, at least in the first stage, simply by providing terms nearer in meaning. An example of some job titles is shown in Figure 2: the hierarchical nature of the titles, and also the existence of some synonyms, is suggested by the numbering scheme, and is more or less self-explanatory.

Figure 2: Examples of job codes and names in French, Flemish and English.

```
91200 cuisinier # kok # cook
91202 chef # chef # chef
91205 chef de cuisine # chef-kok # chief cook
91236 cuisinier de regime # dieetkok # diet cook
91237 cuisinier de cantine # kok grootkeuken # canteen cook
91241 commis de cuisine # keukenhulp # kitchen assistant
91241 commis de cuisine # keukenpersoneel # kitchen staff
91241 commis de cuisine # keukenhulp # catering assistant
91241 aide-cuisinier # hulpkok # assistant cook
91260 second cuisinier # hulpkok # second chef
```

Codes are used as slot fillers in the schema database. This makes the schema neutral with respect to analysis, query and generation languages. For example, when searching for a job, the classification hierarchies inherent in the terminology database allow the user to express general search constraints (e.g. looking for a job as a chef), even though individual jobs are coded for specific types of chef (*pastrycook, pizza chef* etc., and of course in different languages (e.g. *Bakkersgast*).

Although the job titles themselves provide an obvious area of terminology, we handle various other areas of vocabulary in a similar way. There are two criteria for "terminological status" in our system, either of which is sufficient: (i) hierarchical structure, and (ii) standardization. An example of "standardized vocabulary" in our domain is words like *full-time, part-time*, which have an agreed meaning, or adjectives like *essential* as applied to requirements such as experience, or a driving licence. Of more interest perhaps is vocabulary which can be structured, since this provides us with an opportunity to allow more sophisticated searching of the database.

One example is types of establishment, e.g. *hotel, restaurant, cafe, pub* etc. Although such terms do not necessarily figure in recognized terminological thesauri, it is obvious that some structure can be imposed on these terms, for example to enable a user who is looking for a job in an eating establishment to be presented with jobs in a variety of such places. Some hierarchies are trivially simple, for example *full-time/part-time*. A more interesting example is geographical location. Most job ads express the location of the work either explicitly or implicitly in the contact address. But often, these locations are the names of towns or districts, whereas a user might want to search for jobs in a wider area: a user looking for work in Flanders, for example, should be presented with jobs whose location is identified as Antwerp. This is not as simple as it seems however, since the kind of "knowledge" implicated in this kind of search facility is (literally!) "real-world knowledge" rather than linguistic knowledge: short of coding an entire gazeteer on the off-chance that some place-name appeared in a job ad, we must rather rely on the user trials envisaged later in our project to identify the extent to which geographical information needs to be included in the system.

3.3 Lexicon

Not all the vocabulary that the system needs to recognize and handle can be structured in the way just described, so we recognize a second type of lexical resource which, for want of a better term, we call simply "the lexicon". These are words which we often find in job ads, associated with specific slots, which we would like to translate if possible, but which do not have the status of terms, since they are neither structured nor standardized. Examples are adjectives used to describe suitable applicants

271

(e.g. *young, energetic, experienced*), phrases describing the location (e.g. *busy, near the seaside*) or the employer (e.g. *world-famous*) and so on.

Job ads that appear in newspapers and journals can be roughly classified according to their length (short, medium, long) with slightly different lexical and syntactic features accordingly (Alexa & Bárcena, 1992), the details of which need not concern us here. Some of the phrases found in typical job ads serve to signal specific slots (e.g. EM-PLOYER:NAME *is seeking* JOB-TITLE), but these linguistic items do not appear in the lexicon as such. Such elements are regarded as being properly part of the analysis and generation modules, and we describe below how they are handled there.

4 Analysis

The system design permits users offering jobs to submit via an e-mail feed job ads more or less without restrictions. The system converts these texts as far as possible into schematic representations which are then stored in the jobs database. The analysis technique that we have chosen to implement falls into the relatively new paradigm of analogy- or example-based processing. In the following paragraphs we explain the analysis process and discuss our reasons for preferring this over a more traditional string matching or parsing approach.

The input that the TREE system will accept is partially structured, but with much scope for free-text input. One possible way of analysing this would be to employ a straightforward pattern-matching approach, searching for "trigger phrases" such as EM-PLOYER:NAME *is seeking* JOB-TITLE, with special processors for analysing the slot-filler portions of the text. This simple approach has certain advantages over a more complex approach based on traditional phrase-structure parsing, especially since we are not particularly interested in phrase-structure as such. Furthermore, there is a clear requirement that our analysis technique be quite robust: since the input is not *controlled* in any way, our analysis procedure must be able to extract as much information as possible from the text, but seamlessly ignore – or at least allocate to the appropriate "unanalysable input" slot – the text which it cannot interpret.

However, both these procedures can be identified as essentially "rule-based", in the sense that linguistic data used to match, whether fixed patterns or syntactic rules, must be explicitly listed in a kind of grammar, which implies a number of disadvantages, which we will mention shortly. An alternative is suggested by the paradigm of "example-based" processing (Jones, 1996), now becoming quite prevalent in MT (Sumita et al., 1990; Somers, 1993), though in fact the techniques are very much like those of the longer established paradigm of case-based reasoning.

4.1 A flexible approach

In the example-based approach, the "patterns" are listed in the form of model examples. Semi-fixed phrases are not identified as such, nor are there any explicit linguistic rules. Instead, a matcher matches new input against a database of already (correctly) analysed models, and interprets the new input on the basis of a best match (possibly out of several candidates); robustness is inherent in the system, since "failure" to analyse is relative.

The main advantage of the example-based approach is that we do not need to decide beforehand what the linguistic patterns look like. To see how this works to our advantage, consider the following. Let us assume that our database of already analysed examples contains an ad which includes the following: *Knowledge of Dutch an advantage*, and which is linked to a schema with slots filled roughly as follows:

```
SKILLS:LANGUAGE:LANG:nl
SKILLS:LANGUAGE:REQ:"an advantage"
```

Now suppose we want to process ads containing the following texts:

Knowledge of the English language needed. (1)
Some knowledge of Spanish would be helpful. (2)
Very good knowledge of English. (3)

In the rule-based approach, we would probably have to have a "rule" which specifies the range of (redundant) modifiers (asuming our schema does not store explicitly the level of language skill specified), that fillers for the REQ slots can be a past-participle, a predicative adjective or a noun, and are optional, and so on. Such rules carry with them a lot of baggage, such as optional elements, alternatives, restrictions and so on. The biggest baggage is that someone has to write them.

In the example-based approach, we do not need to be explicit about the structure of the stored example or the inputs. We need to recognize *Dutch, English* and *Spanish* as being names of languages, but these words have "terminological status" in our system. If the system does not know *would be helpful*, it will guess that it is a clarification of the language requirement, even if it may not be able to translate it. Furthermore, we can extend the "knowledge" of the system simply by adding more examples: if they contain "new" structures, the knowledge base is extended; if they mirror existing examples, the system still benefits since the evidence for one interpretation or another is thereby strengthened.

4.2 The matching algorithm

The matcher, which has been developed from one first used in the MEG project (Somers et al., 1994), processes the new text in a linear fashion, having

272

first divided it into manageable portions, on the basis of punctuation, lay-out, formatting and so on. The input is tagged, using a standard tagger, e.g. (Brill, 1992). There is no need to train the tagger on our text type, because the actual tags do not matter, as long as tagging is consistent.

The matching process then involves "sliding" one phrase past the other, identifying "strong" matches (word and tag) or "weak" (tag only) matches, and allowing for gaps in the match, in a method not unlike dynamic programming. The matches are then scored accordingly. The result is a set of possible matches linked to correctly filled schemas, so that even previously unseen words can normally be correctly assigned to the appropriate slot.

The approach is not without its problems. For example, some slots and their fillers can be quite ambiguous: cf. *moderate German required* vs. *tall German required* (!), while other text portions serve a dual purpose, for example when the name of the employer also indicates the location. However, the possibility of on-line or e-mail feedback to the user submitting the job ad, plus the fact that the matcher is extremely flexible, means that the analysis module can degrade gracefully in the face of such problems.

5 Query engine

The query engine takes users' specifications of their employment interests to identify those job ads held in the database that match their specification. Input is provided from an HTML form consisting of a number of fields which correspond to job-schema object attributes (e.g. job-title, location etc.). Data entered for any given object attribute is then encoded in the same format used to encode job ad information. Since both (searchable) job ad information and query data are represented in a language-independent format, matches will be made regardless of the language in which the data was entered.

Symbolic case-based reasoning techniques are used to quantify the extent to which users' queries match database objects, allowing the "ranking" of query results.

5.1 Encoding data

Input entered by the user must be encoded using the same method adopted by the analysis module. There are two means by which this can be achieved. One method is to restrict the options available to the user for any given field to a number of possible values for a given object attribute (i.e. provide the user with a Boolean choice). The alternative is to allow users to enter a string which is passed to the terminology module to retrieve the appropriate code. If the string does not return a code, it is considered invalid and the user is requested to enter an alternative.

5.2 Applying case-based reasoning

User-entered information is used to construct a job-schema object which can be considered as the user's "ideal" job. Symbolic case-based reasoning techniques are then applied to quantify the difference between the user's ideal job and jobs held within the database in order to identify those jobs most closely resembling the user's ideal job.

The purpose of using case-based reasoning techniques is to quantify the difference (as a metric value) between any two instances of a job-schema object. That object must be capable of being defined by one or more parameters, with the further requirement that comparison operations upon any two parameter values must yield a numeric value reflecting the semantic difference between the values. Thus, objects can be seen as being located within an n-dimensional parameter space where n is the number of defining parameters of the object.

The parameters which are used to define job ads for TREE are given by the job schema definition, described above. The distance between two values for a specific parameter will be dependent upon the method of encoding but any distance function δ for a given parameter must define the geometric distance between its two arguments (Salzberg & Cost, 1993). That is: a value must have a distance of zero to itself (4), a positive distance to all other values (5), distances must be symmetric (6) and must obey the triangle inequality (7). A further proviso is added that the maximum difference between any two parameter values must be 1, which ensures that all parameters have an equivalent maximal difference (8).

$$\delta(a,a) = 0 \tag{4}$$
$$\delta(a,b) > 0 \text{ if } a \neq b \tag{5}$$
$$\delta(a,b) = \delta(b,a) \tag{6}$$
$$\delta(a,b) + \delta(b,c) \geq \delta(a,c) \tag{7}$$
$$\delta(a,b) \leq 1 \tag{8}$$

For example, a distance function for the job-title parameter (as represented by job-title codes illustrated in Figure 2) could be given by (9),

$$\delta(a,b) = \frac{f(|a-b|)}{n} \tag{9}$$

where a and b are job codes, $f(x)$ returns the number of digits of its argument, and n is the number of digits in the job codes (i.e. $n = 5$). $\delta(a,b)$ evaluates to 1 if the job code arguments differ on the first digit, 0.8 if they differ on the second digit and so on. The job codes are hierarchically ordered so job-title codes that differ over the first digit will refer to greatly different jobs. As such we can see that this parameter distance function would reflect common-sense judgements on the associated job-titles.

The total distance between any two job instances is simply a measure of the distances between indi-

vidual parameter distances and is given by (10),

$$\Delta(A,B) = \sum_{i=1}^{N} \delta_i(a_i, b_i) \qquad (10)$$

where Δ is the instance distance function, δ_i is the distance function for parameter i, N is the total number of parameters by which A and B are defined, and a_i and b_i are the values of parameter i for instances A and B respectively.

Equation (10) provides a measure of the total distance between two instances by summing the distances between all the constituent parameters. Using (10) and a set of parameter distance functions that conform to the properties given as (4)–(8), it is possible to quantify the difference between any job-schema instance held in the database and the "ideal" job-schema object specified by the user. Those parameters for which no value has been specified will exactly match every possible parameter value, and as such the database search is only constrained by those values which users enter.

Since information on job ads is represented in a language-independent format, a search profile in one language will retrieve job ad information entered in any of languages supported. Database queries are conducted by matching the "ideal" job as specified by the user against job-schemas held in the database. The matching process yields a numeric result representing the "distance" between two objects. Identified jobs can then be ranked according to how closely they resemble the user's ideal job. The results of a database query are then fed to the generation module for subsequent presentation in the language specified by the user.

Future plans include increasing the number of fields over which the search can be conducted and permitting users to specify the relative importance of each parameter to the search. The query interface will also keep a record of user "profiles", so that regular users can repeat a previous search the next time they use the system.

6 Generation

The purpose of the TREE generator module is to generate HTML documents in different languages from job database entries (i.e. filled or partially filled schemas), on demand. For several reasons, the approach to generation adopted in the TREE system can be termed "integrated". First, it integrates canned text, templates, and grammar rules into a single grammar formalism. Second, it integrates conditions on the database with other categories in the bodies of grammar rules. Third, it integrates the generation of sentences and the generation of texts and hypertexts in a simple, seamless way. Finally, generation involves just one single, efficient process which is integrated in the sense that

no intermediate structures are created during processing.

6.1 Formalism

In our integrated approach to generation, a grammar rule has the format (11),

$$C_0/S_0 \longrightarrow SS_1, \ldots, SS_n \ \# \ Conditions \qquad (11)$$

where each SS_i has the format C_i, the format C_i/S_i, or the format $[W_1,...,W_m]$. Here, C_i denotes a syntactic category, S_i denotes a semantic value, and W_i a word. The slash symbol "/" is used to separate the syntax from the semantics. The symbol "#" separates the grammar body from a set of conditions on the database. If the set of conditions is empty, the symbol "#", and what follows it, may simply be omitted.

6.2 Canned text, templates, or grammar?

Suppose a system "knows" something, on which we want it to report; suppose it knows that both the Cafe Citrus and the Red Herring Restaurant want to hire chefs, facts which could be captured by the following (logical interface to the) job database:

```
item(e1,x1,y1).
job(y1,91202).
company(x1,'Cafe Citrus').

item(e2,x2,y2).
job(y2,91202).
company(x2,'Red Herring Restaurant').
```

We can imagine setting up our system in such a way that when the system sees facts of this kind, a rule such as the following –

```
s/E -->
  ['Cafe Citrus',advertises,as,vacant,a,
   position,as,chef] #
          {item(E,X,Y),job(Y,91202),
           company(X,'Cafe Citrus')}.
```

– will be triggered, and the system will produce the sentence *Cafe Citrus advertises as vacant a position as chef*. This is a canned-text approach. It is trivial to implement, but the disadvantage is, of course, that we would have to store one rule for each utterance that we would like our system to produce.

As soon as a sentence must be produced several times with only slight alterations, a template-based approach is more appropriate. Let us modify the above rule as follows:

```
s/E -->
  pn/X~name(X,C),
    [advertises,as,vacant,a,position,as,chef]
      # {item(E,X,Y),job(Y,91202),
                    company(X,C)}.
```

The following rule is needed to tell the system that it is allowed to realize the value of the feature <company> as the value itself (i.e. the value is the name of the company).

```
pn/X^name(X,Name) --> [Name].
```

Thus, here too, given the above job database entry, the sentence *Cafe Citrus advertises as vacant a position as chef* can be generated. Furthermore, *Red Herring Restaurant advertises as vacant a position as chef* can be generated as well.

It is not hard to see that the two rules above form the beginning of a *grammar*. Such a grammar may be further elaborated as follows:

```
s/E --> np/X, vp/X^E^A # {A}.
np/X --> pn/X^A # {A}.
np/X --> n/X^A # {A}.
vp/A --> v/X^A, np/X.
pn/X^company(X,Name) --> [Name].
n/X^job(X,91202) --> [chef].
v/Y^X^E^item(E,X,Y) -->
    [advertises,as,vacant,a,position,as].
```

Now, the above sentences, plus many other sentences, may be generated, given appropriate database entries.

Our approach is based on the idea that canned-text approaches, template-based approaches and grammar-based approaches to natural language generation – while they are often contrasted – may in fact be regarded as different points on a scale, from the very specific to the very general. In a sense, *templates are just generalized canned texts, and grammars are just generalized templates*. Indeed, the possibility of combining these different modes of generation has recently been highlighted as one of the keys to efficient use of natural language generation techniques in practical applications (van Noord & Neumann, 1996; Busemann, 1996).

6.3 Processing

Let us now indicate how the rules are meant to be used by the generator module. Traditionally, the process of generation is divided into two steps: generation of message structure from database records (what to say), and generation of sentences from message structures (how to say it). One way of characterizing the integrated approach to generation is to say that we go from database records to sentences in just one step. The process of computing what to say, and the process of computing how to say it, are, in the general case, interleaved processes. The process of generating from a set of grammar rules, given a particular job database entry, will simply involve picking the rules the conditions of which (best) match the entry, and using them to generate a document.

6.4 Generating hypertext

The TREE system provides its output in the form of *hypertext*. This approach has several advantages: first, as argued by (Reiter & Mellish, 1993), the generation of hypertext can obviate the need to perform high-level text structuring, such as assembling paragraphs into documents. "The basic idea is to use hypertext mechanisms to enable users to dynamically select the paragraphs they wish to read, and therefore in essence perform their own high-level text-planning" (Reiter & Mellish, 1993), p.3. Second, but related to the first point, the hypertext capabilities are also a mild form of tailoring to the needs of different users. Users are expected to explore only links containing information that they need.

Hypertext is generated by means of rules that are very similar to the grammar rules described above, but are formulated on a meta-level with respect to sentence/text rules. HTML code "wrappers" can be simply generated around the text. It is fairly straightforward to extend the grammar to other HTML constructions, such as headers, styles, lists, and tables. Using such rules in combination with other rules enables us to produce simple HTML documents, or, if required, quite complex and deeply nested documents incorporating links to other ads, or buttons to expand information, or clarify terminology (e.g. to get a definition of an unfamiliar job-title).

7 Conclusion

The European Union is a loose geo-political organization that has eleven official languages. As such, it is clear that even in a restricted domain such as that of job ads, novel approaches to Language Engineering are required.

In this paper we have described an approach that summarizes ads into a base schema, and then generates output in the desired language in a principled, though restricted way. At first glance, this may look like old-fashioned interlingual MT, but there are two important differences. First, our approach is inherently "lossy", in that not all the information in the input ad may be analysed into the schema. It cannot consequently be included in the generated output. Second, the format of the output can be controlled and customised by the user which means again that the output text is a summary or digest, not necessarily presented in the same order as the original text. For both these reasons, our system cannot be descibed as a "translation system". Nonetheless we believe this approach is capable of giving considerable coverage at a far lower cost and higher quality than that usually associated with MT.

Our approach is not without some disadvantages however: it is well known that a considerable quantity of the semantics of human language is culturally and socially determined. Thus, even though one can map the names of job categories from one language to another, it is not necessarily true that they mean the same thing. So for example, waiters in Spain are expected to serve snacks, whereas in Belgium they do not. There is of course no easy solution to these

275

problems from the Language Engineering point of view: our service must simply advise users to check that the job description in the target country corresponds to their understanding.

Legal constraints are also a significant issue in the area of job advertising. Thus, whilst most countries in the EU have legislation to prevent race and sex discrimination in job advertising, some do not. Thus a Spanish bar can (or could until recently) advertise for *Pretty girls wanted as bar staff*, and *Men wanted to work in the kitchen*. This type of discrimination is illegal in the UK where it would violate Sex Equality Legislation. Thus we must generate non-discriminatory text to avoid running foul of UK law. This clearly shows how practical applications of Language Engineering have to conform in unforeseen ways to the real world.

Our future work will continue to extend the pragmatic approach taken so far. In particular, we are being encouraged to broaden the coverage of our system to include many more employment domains. It remains to be seen what are the consequences of this scaling on what has so far proved to be a simple but effective architecture.

References

Alexa, Melpomeni & Elena Bárcena. 1992. A cross-linguistic study of the sublanguage of short job advertisements for the design of a multilingual text generation system (MEG). CCL/UMIST Report 92/8, Centre for Computational Linguistics, UMIST, Manchester.

Brill, Eric. 1992. A simple rule-based part of speech tagger. In *Third Conference on Applied Natural Language Processing*, Trento, Italy, pp. 153–5.

Busemann, Stephan. 1996. Best-first surface realization. Computation and Language E-Print Archive cmp-lg/9605010. URL http://xxx.lanl.gov/cmp-lg/.

Caldwell, David E. & Tatiana Korelsky. 1994. Bilingual generation of job descriptions from quais-conceptual forms. In *Fourth Conference on Applied Natural Language Processing*, Stuttgart, Germany, pp. 1–6.

EURES. 1989. Communication of the comparison of vocational training qualifications between member states established in implementation of Commission Decision 85/368/EEC of 16th July 1985: Hotel and Catering Industry. *Official Journal of the European Communities* 32, C166, 3 July 1989, pp. 1–56.

Flanagan, Mary. 1996. Two years online: experiences, challenges and trends. In *Expanding MT Horizons: Proceedings of the Second Conference of the Association for Machine Translation in the Americas*, Montreal, Canada, pp. 192–7.

ILO. 1990. *International Standard Classification of Occupations: ISCO-88*, International Labour Office, Geneva.

Jones, Daniel. 1996. *Analogical Natural Language Processing*, UCL Press, London.

Murray, Pamela. 1989. A review of MT policy and current commercial systems in Canada with a view to illustrating the importance of sublanguages in successful MT application. MSc dissertation, UMIST, Manchester.

Onyshkevych, Boyan. 1993. Template Design for Information Extraction. In *Proceedings of the Fifth Message Understanding Conference (MUC-5)*, Baltimore, Md., pp. 19–23.

Reiter, Ehud & Chris Mellish 1993. Optimising the costs and benefits of natural language generation. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambéry, France, pp. 1164–71.

Rumbaugh, James. 1995. OMT: the Object Model. *Journal of Object-Oriented Programming*, 7.8:21–7.

Salzberg, Steven & Scott Cost. 1993. A weighted nearest neighbour algoritm for learning with symbolic features. *Machine Learning* 10:57–78.

Somers, Harold L. 1993. La traduction automatique basée sur l'exemple ou sur les corpus. In *La traductique: Études et recherches de traduction par ordinateur*, Pierrette Bouillon & André Clas (eds), Les Presses de l'Université de Montréal, pp. 149–66.

Somers, Harold L., Ian McLean & Daniel Jones. 1994. Experiments in multilingual example-based generation. In *CSNLP 1994: 3rd Conference on the Cognitive Science of Natural Language Processing*, Dublin, Ireland.

Sumita, Eiichiro, Hitoshi Iida & Hideo Kameyama. 1990. Translating with examples: a new approach to Machine Translation. In *The Third International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Language*, Austin, Texas, pp. 203–12.

van Noord, Gertjan & Günter Neumann. 1996. Syntactic generation. In *Survey of the State of the Art in Human Language Technology* (Ronald A. Cole, general ed.), Chapter 4 (Hans Uszkoreit, ed.). Available at http://www.cse.ogi.edu/CSLU/HLTsurvey/ch4node4.html#SECTION42. To be published by Cambridge University Press.

Vega, José. 1990. Semantic matching between job offers and job search requests. In *COLING-90: Papers presented to the 13th International Conference on Computational Linguistics*, Helsinki, Finland, Vol.1 pp. 67–9.