Natural Language Engineering **1** (1): 1–7

Editorial

Natural Language Engineering was described in the Technical Background Document of the Linguistic Research and Engineering European Programme as follows:

> Linguistic Engineering (LE) is an engineering endeavour, which is to combine scientific and technological knowledge in a number of relevant domains (descriptive and computational linguistics, lexicology and terminology, formal languages, computer science, software engineering techniques, etc.). LE can be seen as a rather pragmatic approach to computerised language processing, given the current inadequacies of theoretical CL (European Commission, 'Linguistic Research and Engineering in the Framework Programme (1991) 1990–1994', p. 7).

This note (in a suitably technical and dry document) was the first mention we encountered of the idea of 'Language Engineering'. However, since then the label has really taken off: conferences, centres of research, and now this journal, have been named around it. At the same time, a good deal of discussion has taken place on what Natural Language Engineering is (or, rather, should be, since we are dealing here with definitions more than with descriptions).

This editorial is an attempt to define the identifying elements of Natural Language Engineering (NLE), and hence define the topic of this new journal, if only in a partial way.

Objective

The principal, defining characteristic of NLE work is its objective: to engineer products which deal with natural language and which satisfy the constraints in which they have to operate. This definition may seem tautologous or a statement of the obvious to an engineer practising in another, well established area (e.g. mechanical or civil engineering), but is still a useful reminder to practitioners of software engineering, and it becomes near-revolutionary when applied to natural language processing. This is partly because of what, in our opinion, has been the ethos of most Computational Linguistics research. Such research has concentrated on studying natural languages, just as traditional Linguistics does, but using computers as a tool to model (and, sometimes, verify or falsify) fragments of linguistic theories deemed of particular interest. This is of course a perfectly respectable and useful scientific endeavour, but does not necessarily (or even often) lead to working systems for the general public.

It will be noted that, in the definition above, the notion of 'constraints to be satisfied' is left rather vague: this is intentional, since constraints vary from those

relating to scale to those relating to robustness, and are all subject to the overriding requirement of cost-benefit. As such, there are many different points of equilibrium which can be reached. We will now examine the possible range of these criteria one by one.

Usability

The first criterion is that there must be a need, i.e. there must be a set of users in the market who would benefit from using the product rather than the current alternatives, and the product is (or may in the future be) usable by those users to satisfy this need. Furthermore they would be prepared to 'pay the price' for it, where 'price' covers not only the cost of the product, but all related costs, such as the inconvenience of learning new techniques, the need to adapt behaviour, the ripple effect on other pieces of technology which would become incompatible or obsolete and so on. If the product is really revolutionary, the target users may not even be aware of their needs, let alone of which technology would satisfy them best: however, even a visionary NL engineer should have a target user base in mind, and a clear idea of why (once educated) the user would want to use the new technology.

For example, a reasonably simple translation system for brand-specific office machine manuals, based on storing and aligning many years of hand translations of previous editions of the manuals (or of ones for similar machines), was recently developed in less than a year by a leading manufacturer for internal use, and resulted in a saving of up to 50 per cent in the effort expended on the production of new manuals for the same line of products. However, a much more sophisticated translation system for intelligence documents, employing word translations, proved unusable, after having absorbed considerably more resources. This is not to say that, in general, small, domain-dependent, surface based solutions are better than large scale, deeper ones under a NLE paradigm: it simply means that the balance of need, production cost, technique currently available, time available, etc. was satisfactory in the first case and unsatisfactory in the second. Usually, being aware of these parameters and their interplay is the first step towards achieving a positive control over them.

From this point of view, successful NL engineering is like successful cooking: there are enormous numbers of wonderful dishes, from very simple staple ones to those for sophisticated entertaining, but they all require a proper balance in their basic ingredients, spices, cooking methods, etc.; unfortunately, there is also an apparently infinite supply of awful concoctions, some of which are very rich, and all unbalanced in some sense. Good NLE is primarily the art of avoiding them.

Cost-benefit

We have already mentioned the cost-benefit analysis that is generally applied (often unconsciously) when judging whether a product is useful; now we will discuss the wider cost-benefit analysis which the NL engineer should apply. In this analysis, the objective must be, as we have seen, the delivery and acceptance by a user base

of the product, and the external requirements of the work must be defined so that they match a balance point for that user base (which means trying to discover, or sometime second guess, the cost-benefit curve applicable to it). This is the basic case: of course, the process may be highly complicated by a range of factors. Among them, are the following:

(a) some of the costs are absorbed not by the final users but by others: typically, government or universities may subsidise part of them, or discount them against other gains, e.g. publications, prestige, education, etc.;
(b) a family of applications is expected to result from one fundamental core effort: e.g. some core systems are nowadays being developed which would, in the plans of their designers, result (after suitable modification and customisation) in applications as far ranging as information extraction, database front ends and translation;
(c) several satisfaction points may be available in the cost-benefit profile of the user base, often separated by large areas of dissatisfaction: e.g. a user may be prepared to buy a cheap, speaker dependent speech recognizer available today, and also an expensive, speaker independent one ready in a year time, but not any of the combinations in between.

Once this analysis has been carried out, the problem becomes one of fitting possible internal requirement sets onto the given external ones. These internal parameters are (at least) the following: resources, scale, techniques, robustness, flexibility, openness, domain dependence, application dependence, efficiency, and implementation issues.

It is important here to remember that over-engineering is a mistake almost as expensive as under-engineering: Henry Ford, who knew a thing or two about engineering processes, used to send people to the car graveyards, to find out which parts looked in good shape when the rest had already rotted away: those parts were deemed to have been over-engineered, and resources were taken away from their manufacturing in order to achieve a suitable quality balance.

Resources

This should follow directly from the understanding of what the user is prepared to pay, plus a subsidy as mentioned above. Again, we are stating the obvious: however, judging from the number of NLP projects left unfinished because the resources available have not been well distributed over the various aspects of the lifecycle, the obvious may indeed be where we need to start from in our area.

Scale

This is an extremely important parameter, which will influence most of the others: e.g. going for a wide coverage grammar, without a suitably robust handling of ungrammatical input, would be a typical example of over-engineering. Similarly, going for a very basic, small scale domain model requires, in most applications, that the necessary information be extracted from somewhere else (e.g. from corpus

statistics, or from large scale general knowledge). Scale is probably the factor with the biggest impact on resources: e.g, it is often possible to solve an efficiency problem with a clever change in the complexity of an algorithm, but usually large scale – be that in grammar coverage, lexicon, domain modelling, general knowledge, statistical data, adaptation, etc. – can only be achieved through painstaking, time consuming work, with the complexity of the task often growing exponentially.

Techniques

There is nowadays a very wide range of tools and techniques available to the NL engineer: traditional, well formalized general linguistic and logic theories; localized theories for specific subsystems (e.g. dialogue); statistical approaches; large corpora; purpose built knowledge bases (e.g. WordNet); heuristic methods; adaptive methods (e.g. evolutionary algorithms or neural nets); and even well publicized ad hoc solutions for hard exceptions. While an individual NL engineer may favour some particular techniques, the paramount consideration should always be what is the best fit for the particular problem. For example, if the researcher is devoted to using a technique independently from the short or long term benefits with respect to other, competing ones, then she or he is really exploring that technique for its own sake, and using NL simply as a test bench: this is, of course, perfectly acceptable, but it would not be NLE as we understand it.

Robustness

It may seem self-evident that a system destined to become a product should be robust in all its aspects, but this is in fact not always the case: it really depends (again!) on the overall balance of the programme. For example, a template scanning application based on initial parsing can often well afford to produce parse trees with wrong prepositional phrase attachment, if that part of the analysis is not usually called upon in the filling of the template slots. It follows that a system which does not excel in any of its parts may still be the best possible combination for a particular task.

Flexibility

A system destined to be used in one set-up only could, conceivably, be built without regard to the issues of flexibility (reuse, maintenance, etc.). However, in practice this will be the case only for systems so small and cheap that the overheads of ensuring flexibility are not worth the gains, or for ones totally built out of existing components (i.e. ones in which the flexibility has already been in-built at a previous stage). In all other cases, flexibility issues must be of great importance: the larger the efforts needed for developing a component, the more worthwhile becomes the search for an existing stand-in, or the need to ensure that the result of such large effort will be used again in future. This principle applies just as well to whole core systems, as it does to components. The central role of reuse and maintenance is well

documented and accepted in software engineering circles nowadays, but the message may still have to make some inroads in the NLP community.

Openness

Openness can be seen as flexibility across various projects and research sites: by making a system open (i.e. compatible with other systems), developers gain access to external resources, but at the same time tie themselves to choices outside their control (and they may also loose some exclusive rights). The tension between these two conflicting interests is a delicate one: an early convergence onto standards may cause serious harm to innovative research, but a complete lack of compatibility forces large groups to reinvent the wheel, and may push smaller ones out of the field altogether. For many years, the pendulum has swung, in our opinion, too far in the direction of lack of compatibility: the situation seems to have improved in recent times.

Domain and application dependence

Like the issue of techniques, the issue of domain and application dependence has been at the centre of heated debates for a long time in the NLP community. A debate that rages for a long time, especially if it does so among people sharing the same goals, is usually a sign of a badly posed question, and this seems to us to be the position in this case. Within a particular context of use either domain dependent or independent approaches may be justified. Equally the use context will define whether an application dependent or independent approach is appropriate. Hence the debate is essentially meaningless. If the final goal is a system able to pass the Turing test on any topic, then clearly domain dependence is not much use; if the goal, however, is to correct the style and content of a technical manual, domain dependence is the core of the issue. Similarly for application dependence: a core system built to support many products will usually be better built with none in mind, rather than as a collection of separate application dependent units.

Efficiency

There are usually two sides to the efficiency of a program: the theoretical one (its complexity), and the practical one. They should both be addressed in a properly engineered NL system. A basic analysis of complexity is essential: an error at that level is often unrecoverable, and it may manifest itself only when large parts of the system have been built. However, the reaction to the total lack of complexity analysis often found in early work in NLP has recently lead to a mythization of the power of complexity analysis. First of all, a complexity analysis is almost always a worst case analysis (as the average case one is often too difficult to identify or calculate): if that worst case is both rare and far outside the normal range, it will be useless, and even misleading. Secondly, even when the worst case is the normal one, complexity is about the behaviour as the input grows toward infinity: in many cases

in NLP, the size of input is bounded, and thus the constants in the estimate, which are irrelevant as far as the order of complexity is concerned, become paramount. An example from another domain is given by quicksort, which has got the best possible complexity of all sorting algorithms: however, for lists shorter than a certain size, other sorting methods are in practice more efficient.

Implementation

We are dealing here with issues such as choice of programming languages, lifecycle models, support tools, etc. A widely repeated cliché is that there are 'languages for AI', 'languages for NLP', etc. We believe this is a false distinction, as it is a false distinction in all other areas of software engineering: the choice of language is determined by a combination of factors (e.g. the scale of the project, the number of developers involved, the languages already known in the group, the complexity of the algorithms, the role and size of data, the importance of interfaces, the availability of support tools, etc.). A good principle in NLE would be to keep an open mind – shop around, remember what the cost-benefit of the choice will be, and don't be influenced by fashion.

Intermediate results

It might be inferred from what we have said above that only finished systems qualify as NLE systems, and thus that only papers on such systems would be welcomed in this journal: this, however, would be a mistaken reading. As long as the NLE principles are adhered to, intermediate results (even highly theoretical ones) can be counted as belonging to the NLE family; conversely, badly engineered system (with respect to the user base) would not qualify, even if they happen to work 'successfully' out of context.

Testing and evaluation

Is the market the only real testing ground for NLE systems? Ultimately, that is indeed the case, in our opinion; however, it would be rather wasteful if that were the only testing means available, considering the time it often takes to develop a system, and the expenses of bringing it to market. Fortunately, there are already intermediate alternatives, and others are being developed. Paramount among the existing ones are the competitions organized by ARPA (Advanced Research Projects Agency) of the USA: MUC, SPREC, TREC and the rest of the Tipster programme. We hope the journal of *Natural Language Engineering* will further promote the active debate on the usefulness and effectiveness of these competitions.

We would also support the development of standard test-sets for the various NLP applications. Discussion on this topic is of fundamental importance, and one of the aims of *Natural Language Engineering* will be to encourage such debate.

Temporary solution?

If we return to the original definition of Language Engineering with which we began this editorial, we see that the authors somehow appeared to apologise for this new direction, explaining that it was necessary 'given the current inadequacies of theoretical Computational Linguistics'. The assumption seemed to be that good engineering (in the sense of technology) is an (inferior) substitute for good science (in the sense of abstract theory), ready to give way once the latter is well developed. This is a neo-positivist philosophical position that has passed its prime long time ago: it is our opinion that, while it is true that good technology may exist even where good theory is lacking (after all, cathedrals were built long before we knew anything about the theory of distributed load, and most of them are still here), it is also the case that good engineering is still needed when the theory is available, not only to put it to use, but ultimately to provide the only defensible test for it.

In conclusion, then, Natural Language Engineering is a new and dynamic field. We hope this new journal will help promote communication within the field and also act as a vehicle for the NLE community to express its distinctive voice to those outside the field.