[Proceedings of the *Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages*, Colgate University, Hamilton, New York, August 14-16, 1985]

ON DEBBUGGING ENVIRONMENT PROPOSED FOR EUROTRA(*) (first draft)

N.VERASTEGUI

Institut de Formation et Conseil en Informatique
27, rue de Turenne - 38000 Grenoble FRANCE

and

Groupe d'Etudes pour la Traduction Automatique -
Université Scientifique et Médicale de Grenoble
BP 68 - 38402 Saint Martin d'Hères FRANCE

I. ABSTRACT

A proposal of external specification of the user environement for the EUROTRA project is presented. The needs of the users and the functions which are necessary for an efficient testing environment are analyzed.

KEYWORDS:
debugging, computational tools for MT, user interface, machine aided translation.

(*) This work has been carried out as part of a contract with the Commission of the European Communities (in the framework of the EUROTRA Research and Development programme) and the CNRS (Centre national de la Recherche Scientifique). The ideas and proposals in this paper are those of the author and not necessarily shared or supported by the Commission, nor are they to be interpreted as part of the EUROTRA design. We are grateful to the Commission and the CNRS for agreement to publish this paper.

II. INTRODUCTION

During the life time of any software, and particulary of a computer aided translation system, the results which are obtained will trigger the modification (or rectification) of certain (linguistic) data, in order to take into account the errors found, the omissions, etc. This debugging process can be summarized as a cycle of TRIAL, DETECTION and MODIFICATION. TRIAL stands for the initiation of the complete or partial translation process, DETECTION means the detection of an error, of an omission, of a possibility to improve this or that work method..., MODIFICATIONS stands for the consequence of this detection on the set of the linguistic data.

The debugging of these data is, then, the result of repeating this cycle until sufficiently accurate results for the kind of texts one wishes to translate are obtained. The functional specifications of the environment monitor must take this remark into account, in order to create maximum automation of this repetitive sequence carried out by a (specialized) linguist.

An efficient testing environment should comprise, in particular:

- the management of the intermediate results, keeping a part of the "history" of the transformations;
- the qualitative and quantitative selection of parts of the corpus of texts and of intermediate results;
- the display of the intermediate results and the control of the details of the display;
- trace tools, with one or more modes of activation;
- the possibility of interactive modifications of the data.

III. SOME CHARACTERISTICS OF DEBUGGING ENVIRONMENTS

The elimination of the errors in a program, whether technical or linguistic, is done in two stages: first by testing the programs in order to detect the errors by studying their effects on the results, followed by debugging the program by the retrieval and modification of the causes of the errors. On the one hand one should, thus, define adequate sets of tests prior to any debugging trial and, on the other hand, understand the program itself (its structure and behaviour) and evaluate the scope of a modification.

The auxiliary systems for debugging or debuggers are well-known in computer science but rarely used, for various reasons:

- they frequently display the wrong kind of information such as, for example, character strings in hexadecimal;
- they are not integrated in the system, for they are generally designed a long time after the programming language;
- they do not use the same syntax as the language for which they have been built.

In conventional computer science, 90% of the errors are found thanks to manual search without the help of the machine, and the remaining errors are detected, in most cases, by "dumps" of the memory. But this last kind of errors are the most hard to find.

It is true that the simple manual checking of the execution of a program with simple data, enables us to detect many of the errors, and this is, moreover, a good method which should be kept.

In the conventional approach of software engineering, the source code is the prime formal representation. In the operational approach, the specifications are both formal and executable: the implementation is derived from the specifications.

A debugger must be adapted to the approach used by the programming language. The communication must, therefore, be established on the level of the syntactic and semantic structures of the language considered and not on that of its implementation, failing which the user will have difficulties to understand the information supplied by the system. Difficulties can also arise in the debugging of optimized programs, in which there is not necessarily a bijective application of the source code towards the object

code. And finally, the solution can vary according to whether the environment is mono-user or multi-user.

The EUROTRA software system is near to the applicative languages and it is implemented with languages of the applicative type <3>. At the core of the applicative languages, we find the functional composition, the undetermined order of execution during the parallel applications and the absence of variables, as the work is done directly on a data structure which is modified with the applications of the functions <1>.

All this considerations has to be taken into account for the design of a testing environment. For instance, a general specification for a debugging environment can be found in <4>. We present here a generalization and adaptation for EUROTRA of what has been realized in ARIANE-78 <2>.

IV. THE DEBUGGER

We propose here the functionalities for the EUROTRA debugger.

It must be possible to give the necessary commands, either in a way which is internal to the models (in a syntactic form to be defined), or in an external way (interactively or at the moment of initiating the execution).

Here, the presentation of the options follows an increasing order of complexity.

1. TRACES

The user must have the possibility to trace the execution of his linguistic application. The simplest mean to achieve this is to supply, at input and at output time of each process(*) (complex or primitive) the initial and final working structure containing, for each of its nodes, the complete content of its decoration, as well as the time

(*) A translation process is decomposed in sub-processes, which can be applied in different ways: sequential, parallel or iterated. Each complex process can be also decomposed in sub-processes until the level of primitive process. A primitive process, by definition, can't be decomposed.

spent (in virtual seconds or in number of machine instructions carried out).

Unfortunately, this solution is satisfactory only for small models working on small structures. In general, the user is "flooded" by the information supplied.

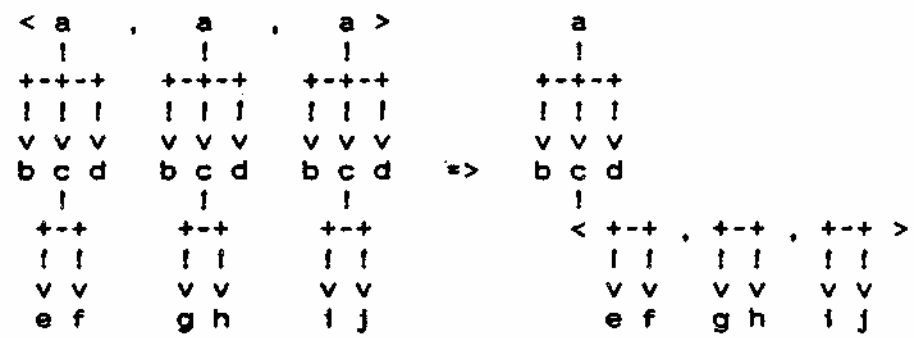
In order to improve this situation, the planning should include parametrizing of traces so as to fix the limitations in time, the localization or the size of the traces to be turned out. We, therefore, propose:

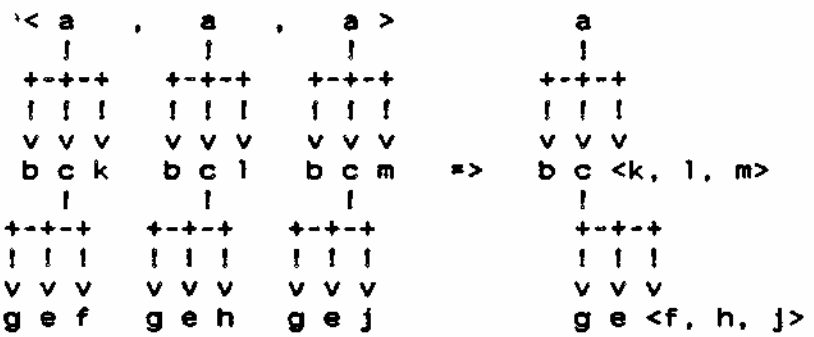
- the possibility of indicating the name(s) of the processes to be traced;
- the possibility of indicating the parts of the decorations to be traced on each node of the structure;
- the possibility to trace only the new or the modified elements;
- the possibility to impose restrictions on the elements to be traced, for example by requesting only the visualization of the nodes possessing a particular decoration, or which verify a condition.

From a linguistic point of view, the structures of the data are forests. The linguist will have no access to the internal data structures. However, if this principle is strictly adhered to, the size of the traces can be considerable and they can be difficult to read.

We therefore propose a factorized writing and/or drawing of the m.a.t. type <5>, which is a more general notion.

If a set of trees have in common (same structure, same decorations) a part of their structure from the roots up till a certain level, all these nodes will be written and/or drawn only once. If two trees have common roots and vary, on the level of their daughters, only in the subsets of the consecutive daughters, the common part will only be written and/or drawn once. These two rules apply recursively to all the sub-trees. For example, (the angular parentheses (<, >) indicate the possible choices):





The EUROTRA internal data structures fully adapt to this notation and its use does not call for any modification of the working structure used by the abstract machine. Moreover, each node of the tree can be marked by a number, and a list can be made of these numbers, followed by the decoration associated with the corresponding node.

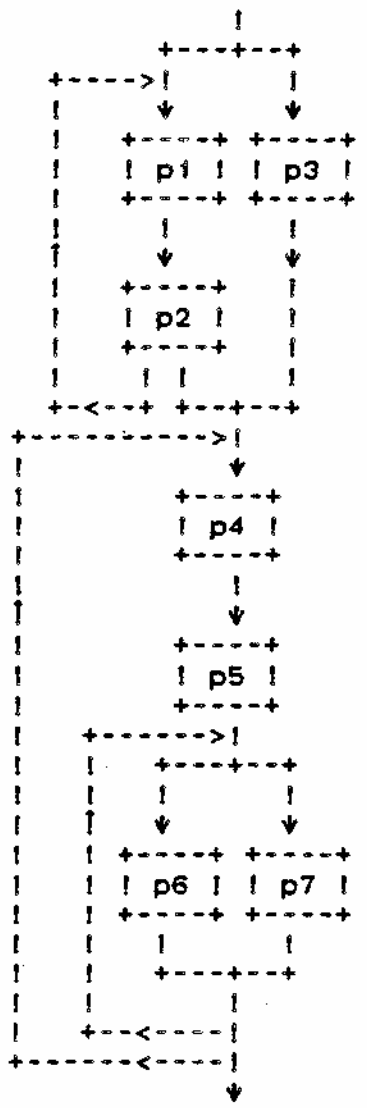
It should also be underlined that this notation applies to partial trees, which is useful when one wishes to extract only part of the data base.

In the previous traces, it is important to identify clearly the process which is concerned. Let us suppose, for example, that we wish to trace the carrying out of the following processes:

```
p12 ::= p11, p9
p11 ::= p10 // p3
p10 ::= (p1, p2)+
p9  ::= (p4, p5, p8)+
p8  ::= (p6 // p7)+
```

The symbols "," "//" and "+" represent respectively the sequentiality, the parallelism and the iteration. Then, we

see that process p12 is equivalent to ((p1, p2)+ // p3), (p4, p5, (p6 // p7)+)+, or, in graphic form:



Although the order of execution of parallel processes is not deterministic, the user should see the traces appear during debugging in the order of writing of the processes. This would considerably facilitate the exploration of the traces. Thus, for the above example, we would have a series of traces presenting the following form (in which the traces at input and at output time are respectively indicated by the characters ' and "):

```
p12' p11' p10' p1' p1" p2' p2" p10" ...
p10' p1' p1" p2' p2" p10" p3' p3" p11"
p9' p4' p4" p5' p5" p8' p6' p6" p7' p7" ...
p6' p6" p7' p7" p8" ...
p4' p4" p5' p5" p8' p6' p6" p7' p7" ...
p6' p6" p7' p7" p8" p9" p12".
```

It is also important to be able to indicate the limits for the traces compared to the levels of the processes. In our example, there are 3 levels of overlapping: p12 at level 0, p11 and p9 at level 1, p10, p8, p3, p4 and p5 at level 2 and the others at level 3. The user can request the traces at level 0, or between the levels 1 and 2, or upwards from 2, etc. If, for example, levels inferior to 3 are requested, traces will be produced such as: p12' p11' p11" p9' p9" p12".

Traces on the level of the grammars and the rules should be planned, but these will depend on the nature of the primitive process.

It would be desirable to have an automatic starting of the traces in case of error. In this way, the size of the traces can be reduced, whilst keeping their power for the precise localizations.

The debugger should enable the production of:

- different statistics;
- the execution frequence of each rule of a model, if required in the form of histograms;
- the running time for each process and/or phase, as well as the percentages compared to the total time from input to output;
- statistics on the use of the dictionary entries, with a view to the improvement of their organization.

2. BREAKS DURING EXECUTION

The system should enable the user to indicate break-points. During the execution and at the moment of each passage through a break-point, the user can ask the debugger in order to know the value of the variables, to

evaluate some expressions, to define or suppress break-points, to continue or terminate the execution.

3. DYNAMIC MODIFICATIONS

When debugging a program, small errors can show up which result in erroneous values which are easily corrected. In general, the program should be modified and recompiled, and the tests should be restarted all over with the new module. We would appreciate the possibility to modify the values controlled by the debugger, so as to continue the test and detect other errors. This is a time-saving means, as the constant shifting from one environment to the other is thus avoided.

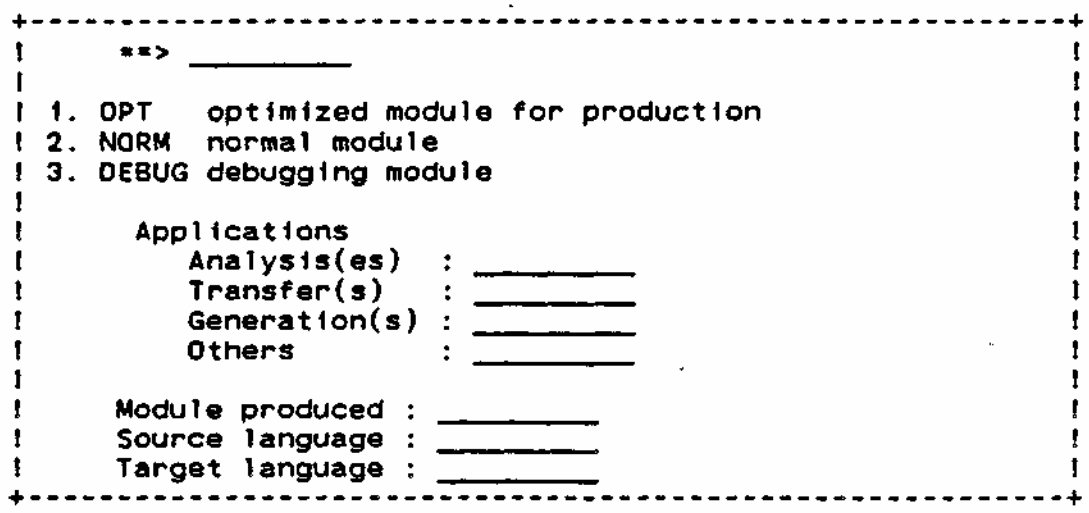
The EUROTRA software system has particular characteristics which complicate the definition of options for modification of the data during debugging: this is the result of the use of parallelism. If modifications are carried out on a parallel branch, the coherence of the working structure on all the other branches cannot be guaranteed. It is proposed (in this precise case) not to permit the execution to be continued past the end of the modified branch.

Anyhow, it should be forbidden to access and modify the structures or informations which have not been defined at the level of the specialized language for linguistic programming.

4. THE MONITOR

The monitor is the means of communication between the user and the EUROTRA MAT system. It is the monitor which carries out the calling of functions of the host operating system, especially concerning the storage of the data and the programs. The final form of the commands will depend on the hardware on which EUROTRA will be implemented and on its operating system.

Suppose that the user has requested the resolution of the external references after a separate compilation and prior to execution, the system could display the following menu which allows the creation of the execution modules:



The user indicates the objective of the module and its components. In the zone of analysis applications, he can indicate several names of applications other than transfer or generation.

In the zone of transfer applications, he can indicate several names of applications other than analysis and generation.

In the generation zone, he can indicate several names of applications other than analysis or transfer.

In the zone "others", he can only indicate the names of applications other than analysis, transfer or generation. In this latter case, the analysis, transfer and generation zones must be empty.

In all cases, the names of the module and the source/target languages are obligatory.

A translation can be carried out in view of three objectives:

- in order to realize the complete translation of one or several texts with the help of a stabilized application (industrial translation);
- in order to realize a partial translation with the help of a stabilized application, with intermediate results at input and/or output time;
- in order to test one or several translation phases.

The choice between these options could be made with the help of the following menu:

```
+-------------------------------------------------------+
!     ==> _________                                     !
!                                                       !
!    module      : __________                           !
!                                                       !
! 1. TRANS  full translation                            !
!                                                       !
!    data       corpus    : __________                 !
!               text(s)   : __________                 !
!  languages    source    : __________                 !
!               target    : __________                 !
!                                                       !
! 2. PTRANS partial translation with intermediate       !
!           working structures                          !
! 3. DTRANS execution controlled by the debugger        !
!                                                       !
!    stages     beginning : __________                 !
!               end       : __________                 !
+-------------------------------------------------------+
```

The source language and target language codes are optional. By default, one can take those indicated during the creation of the module.

V. CONCLUSION

We had proposed a general organization of the system , which would allow, on the one hand, management of the basic EUROTRA system, of the linguistic data and of the texts in all their intermediate stages, and, on the other hand, communication with the users for the preparation, execution and debugging, taking into account the safeguards necessary to guarantee the system's integrity.

The definition of the EUROTRA's specialized language for linguistic programming is not completely finished. The

proposals presented here could be modified as a consecuence of the last choices for the language, especially the proposals on the break-points and on the dynamic modifications should be more thoroughly studied.

VI. APPENDIX E: BIBLIOGRAPHY

1. BACKUS J.,
   "Can programming be liberated from the von Newman style? A functional style and its algebra of programs",
   Communications of the ACM, august 78, vol 21, no 8.

2. BOITET C., GUILLAUME P., QUEZEL-AMBRUNAZ M.,
   "Implementation and conversational environment of ARIANE-78. An integrated system for automated translation and human revision",
   Proceedings COLING82, North-Holland, Linguistic Series No 47, P 19-27, Prague, July 82.

3. JOHNSON R.L., KRAWER S., ROSNER M.A., VARILE G.B.,
   "The design of the kernel architecture for the EUROTRA software",
   Proceedings of Coling84, ACL, 2-6 july, 1984, Stanford U., California, pp. 226-235.

4. SEIDNER R., TINDALL N.,
   "Interactive Debugging Requirements",
   Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on High-Level Debugging, Pacific Grove, California, March 20-23, 1983, pp.9/22.

5. VERASTEGUI N.,
   "Utilisation du parallélisme en traduction automatisée par ordinateur",
   Proceedings of COLING-82, J. Horecky (ed.), pp 397-405, North-Holland Publishing Company, Prague 1982.