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

A CASE STUDY IN SOFTWARE EVOLUTION: FROM ARIANE-78.4 TO ARIANE-85

Ch.BOITET
P.GUILLAUME
M.QUEZEL-AMBRUNAZ

I. TWO SETS OF GOALS

1. INITIAL LINGUISTIC MOTIVATIONS FOR A FIFTH VERSION

1.1. Structure of ARIANE-78.4

Let us briefly recall the structure of the translation process under version 4, which has been presented in detail in many previous papers (6,8).

Logical phase	Physical phase	+/-	Language (SLLP)	Lingware files: Type (number)
ANALYSIS	Morpho-logical AM	+	ATEF	Variables (2) Formats (3) Grammar (1) Dictionaries (1 to 7)
	Structural analysis AS	+	ROBRA	Variables (1) Formats (1) Procedures (1) Grammars (1 to 7)
TRANSFER	Lexical transfer TL	+	TRANSF	Variables (1) Formats (1) Procedures (1) Dictionaries (1 to 7)
	Structural transfer TS	+	ROBRA	as AS
GENERATION	Structural generation GS	+	ROBRA	as AS
	Morpho-logical generation GM	+	SYGMOR	Variables (1) Conditions (1) Formats (1) Grammar (1) Dictionaries (1 to 7)

All phases are mandatory (this is marked by a "+" sign). The lexical information for analysis is essentially

contained in the AM dictionaries, although it is possible to handle special cases (of small classes of words) directly in the AS grammar.

A priority scheme is used in TL to choose, for a given lexical unit, the equivalent(s) given by the highest-ranking dictionary. The priorities may vary from one execution to the other, for example according to the domain being handled. Some dictionaries may even be ignored for a given translation.

1.2. Some linguistically desirable improvements

Numerous discussions with many users have led to the conclusion that some limitations of version 4 were too restrictive and should be removed as early as possible, not waiting for the completion of an entirely new (LISP-based) software. Among them:

1. difficulty of creating a coherent and complete morpho-syntactic indexing scheme for analysis dictionaries, using only ATEF;

2. impossibility to handle during analysis very frequent idioms, such as German verbs with separable particles, without rendering the structural analysis unduly complex;

3. somewhat illogical character of the overall indexing scheme: in analysis, syntactico-semantical properties of the LUs (lexical units) are introduced for each morph, whereas, in transfer and generation, the properties which are used only for generation are given during the lexical transfer. This may cause a duplication of efforts, and a risk of incoherence, in a multilingual setting.

4. restrictive character of the facility provided to change from one decoration type to another, added to the limitation of the size of the decorations ("masks of variables").

1.3. Structure and content of ARIANE-78.5

Those remarks led to a first set of goals, and to the design of a new version called ARIANE-78.5. Let us summarize the main points.

1. ARIANE-78.5 should offer new facilities for lexical processing. To that effect:

   - EXPANS/TRANSFER (EXPANS in short) has been defined, as a new LSPL based on TRANSF;

   - optional "lexical expansion" phases have been introduced.

2. The overall design of a particular translation system should be more modular. To that effect:

   - an extension of the part of the syntax (and semantics) of the SLLPs dealing with the declaration of decoration types and with the passage from one type to another has been defined;

   - it should be possible to construct a structural analyzer as a sequence of transformational systems using the same decoration type.

This led to the following schema for ARIANE-78.5 (see right), where the optional phases are marked by a "-" sign.

2. ADDITIONAL SYSTEM-ORIENTED GOALS

Starting from this set of requirements, it would have been quite possible to satisfy them by just adapting the data structures and slightly modifying some programs.

To take an example, ARIANE-78.4 maintains a file of all LUs (lexical units) of any source (or target) language XYZ (a "language code" such as "ENG" for some version of English), with an indication of the files where they appear (dictionaries, grammars, etc.). It would have been quite easy to modify the structure of this file to allow for more "LU defining files".

However, the fact that there was no urgency in delivering a new version led us to add another set of goals.

Logical phase	Physical phase	+/-	Language (SLLP)	Lingware files: Type (number)
ANALYSIS	Morpho-logical AM	+	ATEF	Variables (2) Formats (3) Grammar (1) Dictionaries (1 to 7)
	Lexical complement AX	-	EXPANS	Variables (1) Formats (1) Procedures (1) Dictionaries (1 to 7)
	Lexical complement AY	-	EXPANS	Variables (1) Formats (1) Procedures (1) Dictionaries (1 to 7)
	Structural analysis AS	+	ROBRA	Variables (1) Formats (1) Procedures (1) Grammars (1 to 7)
TRANSFER	Lexical transfer TL	+	EXPANS	as AX
	Lexical TX complement	-	EXPANS	as AX
	Structural transfer TS	+	ROBRA	as AS
	Lexical TY complement	-	EXPANS	as AX
GENERATION	Lexical GX complement	-	EXPANS	as AX
	Structural generation GS	+	ROBRA	as AS
	Lexical GY complement	-	EXPANS	as AX
	Morpho-logical generation GM	+	SYGMOR	Variables (1) Conditions (1) Formats (1) Grammar (1) Dictionaries (1 to 7)

2.1. Evolution of the SLLPs

First, we wanted to implement the resolution of external entities (such as internal values for the LU strings) differently, and to introduce an explicit link-edit step, instead of maintaining the above mentioned LU files, which are in effect "link maps" of a particular kind.

Second, we felt that the current internal implementation contained too much "handshaking" between the SLLPs and the environment. For example:

- the correspondence between external names of grammars (such as "GR1$AS_ENG" for the structural analysis of ENG) and the internal names of the corresponding files (here, FILASENG_GRAM1_A) is known and used by the compilers and interpreters;

- in the definition of a decoration type for some phases (AS, TL, TS, GM), an implicit reference is made to the definition of the decoration type used in the preceding phase. For example, if "GENDER:=(M,F)." has been defined in AM, it is redefined in AS as "GENDER:=(*).".

2.2. Syntax-directed management of the environment

In ARIANE-78.4, the control structure of the interactive monitor and of the internal supervisor reflects directly the fixed sequence of phases used for translation. As we had already "opened" the system at some critical points (e.g., between AS and TS), in order to experiment with "corrector expert systems", we felt the need to reconsider completely our strategy.

Hence, we added the requirement that the system should in effect be parametrized by the sequence of execution, which should be given in something like a command language. As we will see later on, this idea has led even further.

2.3. User-friendliness: multilinguality and displays

ARIANE-78.4 exists in two versions, French and English. For about 20% of the programs, message files are given as

parameters. For the rest, the programs exist in two versions. We felt that the new release should be made 100% multilingual, by transforming all programs (about 100000 source lines in ASM360, 15000 in PL360. 3000 in PL/I, 25000 in EXEC/XEDIT, 15000 in PASCAL).

When ARIANE-78 was designed, no screens were available, so that the dialogue between the interactive monitor and the user is a sequence of questions and answers. For the production environment (TRAGEN), we had already introduced the possibility to choose on the screen the texts to be translated in a given corpus. Also, the integrated ARIANE help facility ("DET") uses the advantages of the screen. In order for the system to become even more user-friendly, we decided to convert a sizeable proportion of the dialogues to screen-oriented processes.

3. MAIN CHARACTERISTICS OF ARIANE-85

3.1. The SLLPs

The main linguistic processes ("phases") such as AM, AX, AS, etc., are written in one of 4 SLLPs: ATEF, ROBRA, EXPANS/TRANSFER and SYGMOR. EXPANS/TRANSFER is new.

The sublanguage for describing decoration types and the passage between two decoration types has been given the status of a SLLP in its own right, and called TRACOMPL. It still is a sublanguage of each "main" SLLP.

The "handshaking" has been suppressed, so that the compilation of any phase is completely independent of that of any other phase.

A possibility of backtracking in conditional assignments of decorations has been introduced in TRACOMPL, EXPANS/TRANSFER and ROBRA.

Also, the LUs are now considered as strings, and no more as special identifiers for the values of an (implicitly declared) string type (STRING(24), to be precise).

3.2. Describing & using various sequences of linguistic processes

For the checks of coherency and the preparation of linguistic "modules", the user uses an external language for sequence descriptions (ELSD): a set of possible execution sequences is represented as a graph bearing "phases" or explicit conversions of decoration types (TRACOMPL) on the nodes and implicit (REFORM) conversions of decoration types on the arcs. For the execution proper, the sublanguage describing the finite paths of such graphs is used.

The preceding description is subsequently translated into the internal language for sequence descriptions (ILSD), which is a specialized macro-language. The correspondence between external and internal names is computed during this translation.

3.3. The environment

The dialog language (French, English, etc.) is a parameter of the environment. The subenvironments are the same as before (preparation of texts and lingware, tests, morphological checks, production of translations, human revision with on-line computerized dictionary).

The conversational monitor is a combination of a question-answer system and of screen-oriented processes. These processes are implemented using a standard full screen editor. In particular, this enables the system to be driven as. though through a command language, by sending it an appropriate sequence of answers (or commands to the editor), which are stacked in the standard output-restricted deque supported by the operating system (CMS).

3.4. Implementation & usability on new micros

The implementation is still of the same type as explained above. However, the organization of the (virtual) memory during execution has been changed, so that medium-sized applications such as Russian-French (with large grammars and around 10000 LUs, corresponding roughly to 30000 terms) should run in considerably less space than now, where 2.5 Mb are necessary, Including the operating system.

This should make it feasible to run the entire system on an IBM-PC XT/370, and to link it with a translator workstation implemented on a cheap micro, such as TAIM (A.Melby) on IBM-PC (7).

II. THE USER'S POINT OF VIEW

Let us now detail the improvements offered by ARIANE-85, from the user's point of view. They concern all levels of the organization of the lingware:

- definition and use of the linguistic "categories", appearing in the decoration types;
- organization of the lexicon;
- higher level organization of the main processes (phases and subphases).

1. CATEGORIES (DECORATIONS)

1.1. Semantics

For any phase PH, declaring a decoration type amounts in fact to declare two decoration types, called here "DEC1" and "DEC2". Let DEC0 be the decoration type of an input to PH. If PH is AM, the input is a string and DEC0 is empty. Otherwise, the input is a tree decorated on DEC0.

DEC0 is only partially known in PH: DEC1 must be a subtype of DEC0. This constraint is verified by the environment, before any execution is allowed.

If PH is a ROBRA or SYGMOR phase, the whole tree is transformed into DEC2, and then processed by the corresponding automaton (defined by the lingware).

Otherwise, PH is an EXPANS/TRANSFER phase, and the corresponding automaton processes each node N in turn, choosing the image subtree by evaluating conditions on its DEC1 image, called "àN", initializing all image nodes I to its DEC2 image "çN", and then modifying further the DEC2

decoration of each Image node, according to the conditional assignment associated in the dictionary.

The transformation of any node S from DEC0 to DEC2, as defined by the definition of the decoration type, occurs as follows:

1. "reduction" to DEC1, producing a decoration called "àS";

2. "reformatting" (systematic transformation) to DEC2, using the correspondence given in Part A of the syntax (see below), and producing a decoration called "çS". This operation occurs "variable by variable" (attribute by attribute). It may rename the values of the variables (e.g. M to MASC and F to FEM), but cannot modify elementary values.

   Note that the reduction to DEC1 is an implicit reformatting operation. For this reason, it is called a "REFORM" transition. Until now, this is exactly what happens in ARIANE-78.4.

3. "complementary transformation": this is the new extension (TRACOMPL), which appears in Part B in the syntax below. If present, the CVAR expression defines a further transformation of the decoration of each node, expressed as a conditional assignment. This transformation may change values individually, conditionally on values of àS, çS or S. "S" is a name for the current DEC2 decoration of node S, and is initialized to çS.

In EXPANS/TRANSFER, this naming convention (àS, çS, S) for the decorations associated to a node S is used twice. In the definition of the decoration type ("declaration"), it is exactly as we have just said.

In a conditional assignment appearing in an image subtree, "çI" is used to refer to the DEC2 decoration of node S (the processed source node) obtained after executing the complete transformation associated to the definition of the decoration type.

1.2. Sketch of the syntax

```
---------- part A (old) ----------
-EXC- ... -NEX- ... (-ARITH- ...)

---------- part B (new) ----------
((-PROC- ...) (-PRCA- ...) -CVAR- <expression CVAR>)
```

Part A (Declaration and global transposition)

3 forms are possible:

(1) $<variable> : = (<1ist of values>).
<variable> appears in both DEC1 and DEC2, the correspondence between values being positional. Example: $GENDER:=(MASC,FEM).

(2) $$<variable> : = (<list of values>).
<variable> belongs to DEC1, but not to DEC2. The correspondence between values is positional.

(3) <variable> : = (<list of values>).
<variable> belongs to DEC2, but not to DEC1. As a matter of fact, it could belong to DEC0, but to its "unknown" part. For all practical purposes, it is new, and initialized to the associated null value (called "<variable>0", for example "GENDER0").

Hence, the following convention, used in ARIANE-78.4:

"(${$}} <variable> : = (*)."

has become illicit.

Part B (Transformation of values)

By convention, the current node is called S, as above. The identifiers "àS". "çS" and "S" may be used to refer to the various states of the transformation, as explained above.

The parts introduced by -PROC- and -PRCA- are exactly analogous to their counterparts in ROBRA (simple and conditional procedures). A PCIS (internode boolean procedure) with one argument may be called as a PCP (argument free boolean procedure).

For the boolean procedures (PCP and PCIS), we use the notation of ARIANE-78.4. For a given argument node N, not prefixed by "à" or "ç" (indicating DEC1 or DEC2, respectively), the call determines dynamically the decoration type of N.

The part Introduced by -CVAR- is analogous to the expression which may follow "S: çS," in a ROBRA rule of the form:

"S = = S // S : çS. ...".

the only difference being the possible use of àS and çS. S is the only assignable variable.

In the RCA expressions (for conditional assignments), which are analogous to LISP "COND" expressions, the constraint that the last condition be empty has been removed. Semantically, the execution of such an expression (which may be embedded to any depth) occurs in an unary backtracking non-deterministic way; exactly as for the upper level of control in ROBRA (control graph).

1.3. Example

The following example has been taken from the transfer (TL or TX) part of a German-French model.

2. DICTIONARIES

2.1. Semantics of EXPANS/TRANSFER

The new LSPL EXPANS/TRANSFER, or EXPANS in short, has two modes of execution:

- the TRANSFER mode is the same as in TRANSF of ARIANE-78. In the diagram given above for ARIANE-85, it is used only in TL (lexical transfer).

  At execution time, there are two separate LU "variables", corresponding to the "source" and "target" LUs, exactly as if these variables (which are implicitly declared) had been declared as "UL" and "$$UL";

```
                                -EXC-
        ...
$$FS : = (FS1, ... FS24).          ** DEC1.
FS : = (FS1, ... FS20).            ** DEC2.

                                -PROC-

PCP : FS1 = = FS  -E-  FS1.  ** May be used with an
                             argument decoration on DEC1 or DEC2.
PCIS : CD3(XR,XL) = = RL(XR)  -NE-  RLO  -ET-
            (-FS(XR  -E-  FS19  -OU-  FS(XL)  -E-  FS20 -)
            -ET-  FS(çS)  -NE-  FS(S).

   ** At compile time, it is known that XR (the formal
      parameter) may belong to DEC1 or DEC2: the actual
      parameter may be àS, çS or S.
   ** If FS23 and FS24 had been used, XR would be on DEC1,
      and the actual parameter could only be àS.

PAF : FSN20 = =  FS : = FS20.        ** Necessarily on DEC2.
PAF : P3(X)   = = SEM : = SEM(X)  -I-  CONCR  -U-  ABSTR.

                                -PRCA-

SUPP(X2;X1) = = -SI-  FS(X1)  -E-  FS18  -OU-  FS(X1)  -E-  FS19
            -ALORS-  FS(X2) : = FS19.

                                -CVAR-

$$SUPP(S;çS); ** $$SUPP(S;àS) would also be possible.
            -SI-  $CD3(çS,S)  -OU-  $FS1(çS)  -ALORS-  $FNS20.

                                -FIN-
```

- the EXPANS mode is new, and supposes only one LU variable, exactly as if it would be defined as "$UL".

  It is to be used in the "lexical complement" phases, such as AX, AY, TX, TY, GX, GY.

Unlike TRANSF, the underlying automaton allows to:

- use a limited context in the input tree to choose and create the output tree;

- define an action by default, in TRANSFER mode. This action may be used to create dynamically an auxiliary dictionary "0" containing the LUs unreferenced in the

normal dictionaries and the equivalents constructed by the default action.

This is a "defaulter", as in the METAL system, but it is used for translation, and not for analysis, where linguists use the power of ATEF (2) "unknown-word grammar" facility.

The basic idea of the semantics of EXPANS is to see it as a specialization of an (unimplemented) extension of ROBRA (8), let's say ROBRA', which would allow expressions on two decoration types (as in TRACOMPL or EXPANS/TRANSFER) and the selection of right-hand sides of rules in dictionaries.

Let AO be the object input tree. The set of active EXPANS dictionaries, with their priorities, determines one "access expression" (accessor function, in other terminologies), to a rhs of a ROBRA' rule. Everything occurs as if the following grammar were executed, with the only difference that nodes P, G, D, if absent in the lhs, are considered to denote null values in the rhs.

```
                        -GRAM-

GEXPANS(UTH) : E1, E2, E3, E4, E5 ; <-- &NUL.

                        -REGLES-

E1   (S)   P(G,*,S,*,D)    == $$$DICT.   ** "$$$DICT" is used
E2   (S)   P( * ,S,*,D)    = = $$$DICT.  ** here to denote the
E3   (S)   P( * ,S,*,D)    = = $$$DICT.  ** access to the rhs
E4   (S)   P(G,*, S, * )    = =$$$DICT.  ** selected in DICT.
E5   (S)   P( * , S, * )    = =$$$DICT.
```

2.2. Syntax

The procedures (PRC file) have exactly the same syntax as in TRACOMPL:

(-PROC- <PCP, PCIS, PAF>) (-PRCA- <proc. RCA>)

The "formats" (FCP and FAF files, used to define constant decorations on DEC1 and DEC2, respectively), are exactly as in ARIANE-78.4.

The syntax of the dictionaries is augmented:

- choice conditions may be internode expressions or procedures (CIS or PCIS) on "àS". "àP", "àG" and "àD";

- there is a new, special (and optional) dictionary "0", where the user may write his default action, as the first item, using a special form of the syntax:

  <empty (no LU string)> = = // <default action>.

  The default action is made of:

  - an expression for creating a target LU string;

  - a normal <tail of assignment>.

- in assignment expressions, two cases are distinguished:

  - the image subtree is implicit, or (equivalently) given as "S". "S" may be assigned, and "S", "çS", "àS", "àP", "àG", and "àD" may be used for tests or as source of values;

  - the image subtree is explicit. Each image node "I" may be assigned, and "I", "çI", "àS", "àP", "àG", and "àD" may be used for tests or as source of values.

Example and use

```
= = // '?!' !! UL !!'!?', *FT1, $PC3.
```

If UL(S) = 'XYZT' (undefined!), the expression:

"UL(S) : = '?!XYZT!?', *FT1, $PC3. "

will be executed, and the item:

```
'XYZT'    = = // '?!XYZT!?', *FT1, $PC3.
```

will be added to dictionary 0. "!!" denotes of course the concatenation of strings. Incidentally, this is the only place in ARIANE-85 where this operation may be used on LUs.

3. ORGANIZATION OF THE PROCESSES (SUB)-PHASES

3.1. Subphases

A phase is defined as a collection of files written in a certain SLLP. A subphase is any subcollection, usually augmented with an ordering, which gives rise to a coherent and executable lingware.

For any EXPANS phase, it is possible to define up to 7 dictionaries, and to use any subset of them, with a priority scheme, expressed by a very simple expression such as: "4 5 2 6 3 1". According to the domain of a text, it is possible to obtain more specific translations, or even sets of properties ("to code for" appears nowadays only in biochemistry and genetic engineering, for example).

For any ROBRA phase, it is possible to define a set of up to 7 transformational systems, written on separate files, but using the same decoration type. At execution time, the user may choose to use (some of) them in an appropriate sequence.

This division may be used to address several kinds of texts with the same lingware:

- if there are several input formats, it is possible to normalize them, using different transformational subsystems, at the beginning of the structural analysis;

- if there are several typologies, the subsystems using typology-specific heuristics to resolve particular problems (ambiguity, anaphora,...) may also be kept as separate modules;

- in applications where there is a sufficient proximity of the two languages for a simplified linguistic strategy to be cost-effective, the main part of the structural analysis may exist in two versions;

several kinds of transfer and generations may be developed simultaneously, for different purposes, such as normalization within the same language, or prosodic generation.

In ATEF, it is also possible to develop several alternative grammars, as was already the case for SYGMOR.

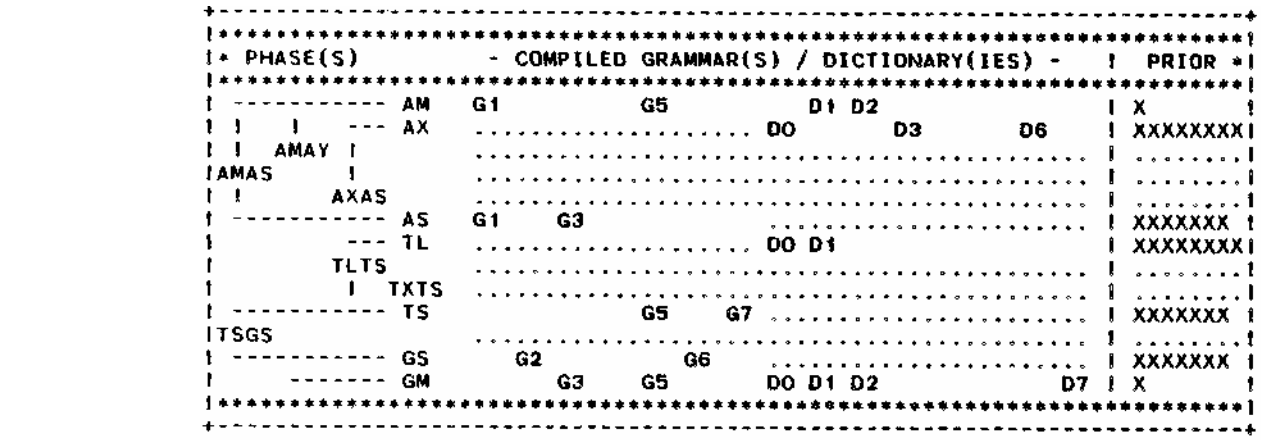
3.2. Phases

The introduction of optional phases AX and AY should make it possible to incorporate the lexical knowledge in a more modular fashion. For example, AM may be used to associate lemmas and strictly morphological information with the morphs. Then, AX may be used to relate the lexical units to the lemmas, and to give them syntactico-semantic information, such as valencies, argument frames and semantic restrictions on the arguments. AY may then add Information about the syntactico-semantic behavior of idioms whose components are separable or flexional, such as compound predicates or German verb-particle constructions.

Other optional EXPANS phases have been included in the above diagram. The most important is GX, the most natural use of which is to index all properties of target lexical units not necessary at transfer time -- that is, almost all. Until now, all lexical target information (not including the strictly morphological information) had to be included in all TL dictionaries for all language pairs with the same target language.

3.3. Specifying a given organization under the new monitor

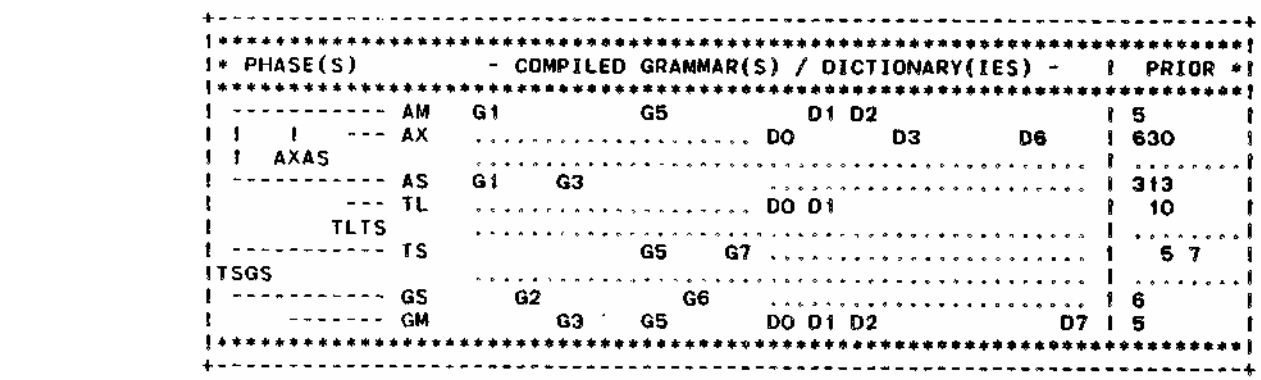
Let us now give an example of a screen prepared by the monitor.



Two grammars and three dictionaries exist in AM.

The TRACOMPL AMAY phase has been compiled, but not yet the AY phase, which may even be absent.

The TXTS TRACOMPL phase is compiled, but not the TX phase.
The user now edits the file on screen, in order to choose a particular sequence of phases, and arrives at the following screen:



Grammar 5 has been chosen.
Priorities: search in DIC6, then in DIC3, and then in DICO.
The AS subphase will execute G3, G1, and G3 again.

The following screens are used to select all trace or output parameters.

```
+-------------------------------------------------------------------------+
!*************************************************************************!
!*        !        -EXECUTION PARAMETERS-          ! **   (AS A REMINDER)  **!
!*        !         ------------------             ! **     DON'T MODIFY!  **!
!*PHASE(S)! TR.. !  RESG  ! TPS ! INTER ! MOINC ! **   GR.    !   DIC.    **!
!*************************************************************************!
!* AM     ! XXX  ! XXXXXX ! XX  !  X X  !  XX   ! ** 5       ! 12        **!
!* AX     !  N   ! XXXXXX ! XX  !  . .  !  ..   ! ** ....... ! 630       **!
!* AXAS   !  N   ! XXXXXX ! XX  !  . .  !  ..   ! ** ....... ! ........  **!
!* AS     ! XXXX ! XXXXXX ! XX  !  . .  !  ..   ! ** 313     ! ........  **!
!* TL     !  N   ! XXXXXX ! XX  !  . .  !  ..   ! ** ....... ! 10        **!
!* TLTS   !  N   ! XXXXXX ! XX  !  . .  !  ..   ! ** ....... ! ........  **!
!* TS     ! XXXX ! XXXXXX ! XX  !  . .  !  ..   ! ** 57      ! ........  **!
!* TSGS   !  N   ! XXXXXX ! XX  !  . .  !  ..   ! ** ....... ! ........  **!
!* GS     ! XXXX ! XXXXXX ! XX  !  . .  !  ..   ! ** 6       ! ........  **!
!* GM     ! YY   ! YY  YY ! XX  !  . .  !  ..   ! ** 5       ! 0127      **!
!*************************************************************************!
+-------------------------------------------------------------------------+
```

Screen before the selection.

```
+-------------------------------------------------------------------------+
!*************************************************************************!
!*        !        -EXECUTION PARAMETERS-          ! **   (AS A REMINDER)  **!
!*        !         ------------------             ! **     DON'T MODIFY!  **!
!*PHASE(S)! TR.. !  RESG  ! TPS ! INTER ! MOINC ! **   GR.    !   DIC.    **!
!*************************************************************************!
!* AM     ! TIQ  ! TIQLAS ! N   ! 3GSI  !  TI   ! ** 5       ! 12        **!
!* AX     !  N   !  N     ! T   ! ....  !  ..   ! ** ....... ! 630       **!
!* AXAS   !  N   !  TG    ! I   ! ....  !  ..   ! ** ....... ! ........  **!
!* AS     ! TIQC !  A     ! TI  ! ....  !  ..   ! ** 313     ! ........  **!
!* TL     !  N   !  S     ! N   ! ....  !  ..   ! ** ....... ! 10        **!
!* TLTS   !  N   !  AS    ! T   ! ....  !  ..   ! ** ....... ! ........  **!
!* TS     ! N    !  IQAS  !  I  ! ....  !  ..   ! ** 57      ! ........  **!
!* TSGS   !  N   !  IQD   !  N  ! ....  !  ..   ! ** ....... ! ........  **!
!* GS     ! TR   !  IDS   !  N  ! ....  !  ..   ! ** 6       ! ........  **!
!* GM     ! T    !  ES  I !  N  ! ....  !  ..   ! ** 5       ! 0127      **!
!*************************************************************************!
+-------------------------------------------------------------------------+
```

Screen after the selection.

Now, the above files are used to generate a program in the ELSD (see Part III). Then, the ARIANE monitor checks it for consistency and translates it into an equivalent ILSD program.

From the user's point of view, all improvements incorporated in ARIANE-85 are upward-compatible, with the minor exception of the definition of decoration types, where the "*" must be replaced by the corresponding list.

In the future, the modifications in the very structure of the implementation should make it more attractive to link lingwares written in ARIANE-85, and incorporating (volontarily) only linguistic knowledge, to other (expert) systems relying more on meta- or extra-linguistic knowledge [11, 12].

III. THE IMPLEMENTOR'S POINT OF VIEW

The internal level of the software comprises the processors for the SLLPs (compilers, interpreters, loaders, supervisor). At this level, the logical structure of the specialized data-base is ignored.

The external level includes the interactive monitor and the data-base management utilities, which of course make use of internal level routines.

1. THE SLLPS

A main principle of the evolution has been to reuse existing modules, as far as possible. As a matter of fact, this has been done to a large extent. Low-level programming may be structured, and, conversely, programming in a high-level language is not a guarantee of structured programming!

1.1. Compilers

The compilers for ATEF, ROBRA and SYGMOR have simply been adapted, in order to produce lists of external references (the LU string values), instead of accessing and modifying a shared LU file (see above). Parametrizing the messages by the dialog language was not too difficult, due to the modular construction of the compilers.

A new compiler for TRACOMPL has been built. It includes some parts from the old compiler for “variable decorations”, and from the old ROBRA compiler.

In the same manner, the new EXPANS/TRANSFER compiler reuses some parts of the old TRANSF compiler. The main differences are that:

- the use of a limited context leads to an important extension of the rhs syntax;
- the hash-code is constructed differently, because the internal values of the LU strings are not known any more at compile time. At loading time, the internal values are used to finish the construction of the hash-code.

1.2. Interpreters

The interpreter for TRACOMPL uses the previous algorithm for the reduction and reformatting operations. For the complementary transformation, however, a new one has been designed, by extending the existing evaluator of expressions and procedures on decorations (RBEVMSK). The extension introduces the possibility of back-tracking in conditional assignments.

In all interpreters, the status of the lexical units has changed. In ARIANE-78.4, there are in effect two sets of LUs during a given execution, the static set (made of all values appearing in the lingware), and the dynamic set (created in AM by the subgrammar for unknown words). In some cases, the same string may correspond to 2 different LUs, one static and the other dynamic, which have 2 different internal values. An internal value is a two-byte signed integer.

In ARIANE-85, the LU strings are the real values. The internal integer values are dynamically associated to the strings, and may vary from one phase to the other.

The integer assigned to a given string is simply the rank of the string in the lexicographic ordering of the set of all LU strings known at a given moment. This ordering is obtained by a classical and efficient merge of the (already sorted) lists of LU strings produced by the compilers or contained in the internal representations of the decorated trees associated to the units of translation.

But for the handling of the LUs, the interpreters for the grammars (ATEF, ROBRA, SYGMOR) have not changed at all.

1.3. Loaders

In ARIANE-85 as in ARIANE-78.4, it is necessary to load the appropriate compiled programs (interpreters and utilities) and the compiled lingware before any execution is possible. Then, modules (binary core images) may be created, and used later as binary translation programs.

In ARIANE-78.4, such a module contains both a software and a lingware part. In ARIANE-85, the loaders have been

modified in order to be able to generate separate modules for the software and the lingware parts.

A further modification is the inclusion to a link-edit step for the LUs: the loaders store the internal value of each UL in the compiled (and loaded) lingware code, at the appropriate locations.

Also, a special kind of loading operation produces the internal tables for the implicit "reduction" operations (REFORM phases, or "empty" transitions in the graph of possible sequences). This makes it possible for the data-base monitor to check for consistency before starting any execution.

2. DB MANAGEMENT AND SEQUENCING: EXTERNAL LEVEL

2.1. A specialized data-base

As has been said before, the interactive monitor (ARIANE for short) manages a specialized data-base of texts and of lingware files, in the user space. Of course, it uses resources from the system space, always accessed in read-only mode.

In the current state of ARIANE-85, the structure of the lingware mirrors the diagram given at the beginning of this paper for ARIANE-78.5. In the future, it might be possible to use a description of possible sequences of phases to generate an appropriate description of the data-base.

The data-base is organized around external notions such as source (or target) language code, or "elements" of (sub)-phases in a given SLLP.

For any SLLP, the data-base knows which are the possible "elements", their type, and their mutual dependencies. These elements are stored in the above mentioned "lingware files". For example, in TL (EXPANS/TRANSFER in TRANSFER mode), we have:

- DV, for the "declaration of variables" (definition of the decoration type);
- PRC, for the procedures used to choose between several image subtrees;

- FCP, for the formats on DEC1 (used in conditions on the source);

- FAF, for the formats on DEC2 (used as source of attribute values in assignments);

- DIC1 to DIC7, for the dictionaries.

As in ARIANE-78.4. the compilation of the elements is separate. ARIANE knows, however, that DV must be compiled before FAF, and PRC before any DICn. Everything happens as if DV, PRC, etc. were explicitly "imported" in each DICn.

This knowledge is used whenever an element is modified: all depending elements are automatically decompiled. In the same manner, when the user asks for the compilation of a phase (or of a subphase), ARIANE knows the appropriate order in which to compile the elements.

The evolution from ARIANE-78.4 to ARIANE-85 has made it possible to suppress all dependencies between phases. Before, any modification in the DV of AS caused AS and all transfers from the considered source language code to be decompiled. Generation phases were left untouched, because, in ARIANE-78.4, the values of all variables must all be redefined in GS, using the "$" convention (see Part I).

ARIANE also commands the generation of lingware modules and stores them in the user space for use in the testing or production environments.

All LU-handling facilities of the data-base have been rewritten, according to the new implementation of the LUs. For example, it is possible to ask for a list of the LUs appearing in DIC1, DIC2 and DIC5 of TL, and not in DIC6 and in GRAM5 of AS.

2.2. An External Language for Sequence Descriptions (ELSD)

Two examples have been given above. This ELSD has been designed by the third author with the aim to mix the unavoidable linear form with the more conspicuous graphical form. The graph appears in the left part of the description, and the corresponding elements or parameters are given linearly in the right part.

The files which appear on the screen are normal files. After checking, they are transformed into a program in the ELSD. For the example we have used before, we obtain the following program, which will then be translated into the ILSD (see below).

```
LANGUAGE(S)  (ENG-FRA)

                -EXECUTION PARAMETERS-                          -PROGRAM TRACES-
                ----------------------                          ----------------
*PHASE(S)!   GR:   !   DIC:    ! TR.. !  RESG   ! TPS ! INTER ! MOINC ** TCH !   TSUP    ! MAPZ ! TROB **
*********************************************************************************************************
* AM      ! 5       ! 12       ! TIQ  ! TIQLAS ! N   ! 3GSI  !   TI   **  N   ! NNNN      !  NN  ! .... **
* AX      ! ....... ! 630      ! N    ! N      ! T   ! ....  !   ..   **  N   ! NNNN      !  NN  ! .... **
* AXAS    ! ....... ! ........ ! N    ! TG     ! I   ! ....  !   ..   **  N   ! NNNN      !  NN  ! .... **
* AS      ! 313     ! ........ ! TIQC ! A      ! TI  ! ....  !   ..   **  N   ! NNNN      !  NN  ! NNON **
* TL      ! ....... ! 10       ! N    ! S      ! N   ! ....  !   ..   **  N   ! NNNN      !  NN  ! .... **
* TLTS    ! ....... ! ........ ! N    ! AS     ! T   ! ....  !   ..   **  N   ! NNNN      !  NN  ! .... **
* TS      ! 57      ! ........ ! N    ! IQAS   ! I   ! ....  !   ..   **  N   ! NNNN      !  NN  ! NNON **
* TSGS    ! ....... ! ........ ! N    ! IQD    ! N   ! ....  !   ..   **  N   ! NNNN      !  NN  ! .... **
* GS      ! 6       ! ........ ! TR   ! IDS    ! N   ! ....  !   ..   **  N   ! NNNN      !  NN  ! NNON **
* GM      ! 5       ! 0127     ! T    ! ES  I  ! N   ! ....  !   ..   **  N   ! NNNN      !  NN  ! .... **
*********************************************************************************************************
```

2.3. Using a general-purpose editor and not a screen handling facility

The decision to use XEDIT, a general-purpose screen-oriented editor, may be contested. Why not use a general menu-generating facility? The answers are the following:

- no additional external tool is used. Improving the maintainability of the system;
- this technique has been successfully used for the VISULEX interface and for the THAM revision environment;
- the same editor is used to modify the lingware files;
- as mentioned before, this makes it possible to drive the system in disconnected mode, by sending to it answers or editor commands, such simulating a human operator.

Of course, it would be even nicer to use a syntactic editor, tailored to the SLLPs and to the ELSD. But this would have been a project in its own right.

3. SEQUENCING LINGUISTIC PROCESSES: INTERNAL LEVEL

It would have been possible to command the execution of a sequence of phases directly from the interactive monitor, by programming it in EXEC2 (CMS Shell). But it would then have been impossible to avoid the constant use of files for transmitting intermediate results (tree structures) between phases.

3.1. An Internal Language for Sequence Descriptions (ILSD)

Hence, an internal langage for sequence descriptions

ILSD) has been designed and implemented by the second author. The idea is to express a set of possible sequences of phases, with the choice of subphases, as a graph, where:

1. each node corresponds to a "main" phase (in ATEF, ROBRA, EXPANS or SYGMOR), or to an explicit transformation phase (TRACOMPL);

2. each arc corresponds to an implicit transformation of the decoration type (REFORM).

Let use give an example corresponding to the graph described above at the external level.

```
ç  Fragment for 5 phases (see example above) of an
ç ILSD program for a graph of possible sequences.
ç
ç  ATEF(1) -  EXPANS(4) -  TRACOMPL(6) -  ROBRA(8).
ç       ou ATEF(1) -  TRACOMPL(9) -  ROBRA(S).

ç  (1) ---- 3->-- (4) ---- 5->-- (6) ---------- 7->--   (8) *.
ç   !                                                   !
ç   !------ 2 ->--~------ (9)------------------ 10 ->----!

ç Node (1), AM phase (ATEF)

          STEP = 1 ;                              çNode 1 (STEP)
          TYPHA2 = ATEF ;                         çLSPL
          NOMPHAZ = AM ;                          çName of phase
          FIFAZ  = CHG01ENG AM;                   çGeneral descriptor
          FSTEP ;

ç Transition AM (1) to AMAS (9) ---- REFORM operation

          SKIP = 2 ;                              çArc 2
          NOMPHAZ = REF2 ;                        çName of phase
          INDPHAZ = --- REFORM: AM->AMAS --- ; çHeading
          DET = I ;                               çTrace
          FITRANS = CAMXXENG RAMAS;               çTranslation table
          FISOR   = FAMXXENG LAMAS;               çTable for files
          FIZON   = FAMXXENG ZAMAS;               çMap of zones
          MODULE  = FAMXXENG MAMAS;               çLinguistic module
          LS = ENG ;                              çLanguage code
          SUISTEP = 9 ;                           çDestination of the arc
          FSKIP ;
```

```
ç Transition AM (1) to AX (4) --                REFORM operation

          SKIP = 3 ;                            çArc 3
          NOMPHAZ = REF3 ;                      çName of phase
          INDPHAZ = --- REFORM: AM->AX  ---------- ; çHeadlng
          DET = I ;                             çTrace
          FITRANS = CAMXXENG RAXXX;  çTranslation table
          FISOR   = FAMXXENG LAMAS;  çTable for files
          FIZON  = FAMXXENG ZAMAS;   çMap of zones
          MODULE  = FAMXXENG MAMAS; çLinguistic module
LS = ENG ;                                      çLanguage code
ç    no SUISTEP: last arc from node 1, goes to next node: 4
FSKIP ;


ç Node (4), AX phase (EXPANS)

          STEP = 4 ;                            çNode 4
          TYPHAZ = EXPANS ;                     çLSPL
          NOMPHAZ = AX ;                        çName of phase
          FIFAZ   = CHGO1ENG AX;                çGeneral descriptor
          FSTEP ;


ç Transition AX (4) to AXAS (6)          -- REFORM operation

          SKIP = 5 ;                            çArc 5
          NOMPHAZ = REF5 ;                      çName of phase
          INDPHAZ = --- REFORM: AX->AXAS --- ; çHeading
          DET = I ;                             çTrace
          FITRANS  = CAXXXENG RAXAS;  çTranslation table
          FISOR       = FAXXXENG LAXAS;  çTable for files
          FIZON       = FAXXXENG ZAXAS;  çMap of zones
          MODULE  = FAXXXENG MAXAS; çLinguistic module
          LS = ENG ;                            çLanguage code
          FSKIP ;



ç Node (6), AXAS phase (TRACOMPL)

          STEP = 6 ;                            çNode 6
          TYPHAZ    = TRACOMPL                  çLSPL
          NOMPHAZ = AXAS ;                      çName of phase
          FIFAZ       = CHGO1ENG AMAX ;  çGeneral descriptor
          FSTEP ;
```

```
ç Transition AXAS (6) to AS (8)                    -- REFORM operation

          SKIP = 8 ;
          NOMPHAZ = REF8 ;                         çName of phase
          INDPHAZ = ---- REFORM: AXAS->AS - --------- ; çHeading
          DET = I ;                                çTrace
          FITRANS  = CAXASENG RASXX;               çTranslation table
          FISOR    = FAXASENG LASXX;               çTable for files
          FI20N    = FAXASENG ZASXX;               çMap of zones
          MODULE   = FAXASENG MASXX;               çLinguistic module
          LS = ENG ;                               çLanguage code
          VAR2 = -------- ----- ;                  çVariables
ç no SUISTEP: last arc from node 6, goes to next node: 8
          FSKIP ;


ç Node (8), AS phase (ROBRA)

          STEP = 8 ;                               çNode 8
          TYPHAZ   = ROBRA ; .                     çLSPL
          NOMPHAZ = AS ;                           çName of phase
          FIFAZ    = CHGO1ENG AS;                  çGeneral descriptor
          FSTEP ;

        DSEQ ;    çGoing back to                   node 9 ("hanging")


ç Node (9), AMAS phase (TRACOMPL)

          STEP = 9 ;                               çNode 9
          TYPHAZ   = TRACOMPL                      çLSPL
          NOMPHAZ = AMAS ;                         çName of phase
          FIFAZ    = CHGO1ENG AMAS ;               çGeneral descriptor
          FSTEP ;


ç Transition AMAS (9) to AS (8)                  -- REFORM operation

          SKIP = 10 ;                              çArc 10
          NOMPHAZ = REF10 ;                        çName of phase
          INDPHAZ = --- REFORM: AMAS->AS --- ; çHeading
          DET = I ;                                çTrace
          FITRANS  = CAMASENG RASXX;               çTranslation table
          FISOR    = FAMASENG LASXX;               çTable for files
          FIZON    = FAMASENG ZASXX;               çMap of zones
          MODULE   = FAMASENG MASXX;               çLinguistic module
          LS = ENG ;                               çLanguage code
          SUISTEP = 8 ;                            çDestination of the arc
          FSKIP ;
          END ;                                    çEnd of the graph
```

```
*     CHG01ENG AM -- general descriptor for AM
TYPHAZ    = ATEF                                  çSLLP
NOMPHAZ = AM                                      çName of phase
INDPHAZ  = --- PHASE DE MORPHOLOGIE ---           çTrace heading
FICOR      = CHFICENG AM                          çFiles to load
PARAM     =  CHPARENG AM                          çParameters
FISOR      = FAMXXENG LDONG                       çTable of files
FIZON      = FAMXXENG LZONG                       çWork areas
MODULE   = FAMXXENG MODO1                         çLing. module
*       CHF01ENG AM -- files to load for AM
VARM       = FCPAMENG VIN2M                      çDVM
VARS       = FCPAMENG VIN2S                      çDVS
MASO       = FCPAMENG FINTM                      çFTM
REFORM   = FCPAMENG FRFIM                        çReferences FTM
FORMATS = FCPAMENG FITTS                         çFTS
REFORS    = FCPAMENG FRTIS                       çReferences FTS
GRAM       = FCPAMENG GINT5                       çGRAM
REFREG    = FCPAMENG GRFR5                       çRef. rules
REFULG    = FCPAMENG GULG5                       çRef. LU
DICT1       = FCPAMENG DITT1                     çDIC1
TABDIC1   = FCPAMENG DTIN1                       çDIC1 (h-code)
REFULD1  = FCPAMENG DULG1                        çDIC1 (LUs)
DICT2       = FCPAMENG DITT2                     çDIC2
TABDIC2   = FCPAMENG DTIN2                       çDIC2 (h-code)
REFULD2  = FCPAMENG DULG2                        çDIC2 (LUs)
NUMUL     = FCPAMENG ULPHG                       çLUs for AM
*       CHP01ENG AM -- parameters for AM
LG            = ENG                          çSource language
GD           = G                             çDirection of analysis: G=left
TT            = A                            çWithout "homophrases"
DICT         = 1,2                           çDictionaries used
MU           = N                             çLevel of interaction
GS           = N                             çOutput of the graph
DT           = N                             çTrace of loading
DETSUPG  = NNNN                              çTrace of supervisor
MAPZ        = NN                             çMap of zones
*       CHGO1ENG AS -- general descriptor for AS
TYPHAZ    = ROBRA                            çType of phase
NOMPHAZ = AS                                 çName of phase
INDPHAZ  =    - ---------- PHASE "AS" ----    çTrace heading
FICOR      = CHFICENG AS                     çFiles to load
PARAM     = CHPARENG AS                      çParameters
FISOR1     = FASXXENG LDON1                  çGR1: Table of files
FIZON1     = FASXXENG LZON1                  çGR1 : list of zones
MODULE1  = FASXXENG MODO1                    çGR1 : module
FISOR3     = FASXXENG LDON3                  çGR3: table of files
FIZON3     = FASXXENG LZON3                  çGR3: list of zones
MODULES  = FASXXENG MODG3                    çGR3: module

            etc.
```

The ILSD is a specialized macro-language. For reasons of efficiency, it has been made considerably simpler than the EXEC2 language of CMS used at the external level, and its interpreter has been implemented directly in assembler.

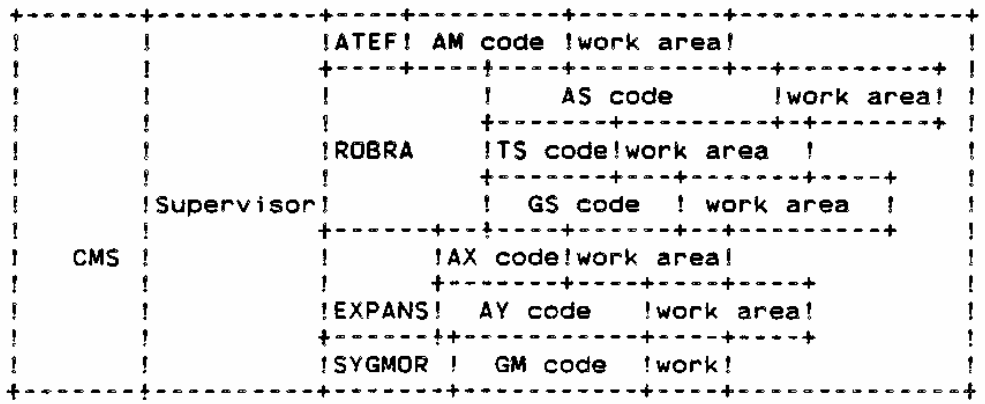
3.2. A generalized supervisor

The new supervisor has been designed and implemented with the aim:

- to decrease the size of the necessary virtual memory;
- not to augment the use of external memory;
- to use separate software and lingware modules;
- not to use any new external tool (such as a screen-handling facility).

Those three aims express the desire to satisfy the constraints of the new IBM-PC XT/370 or AT/370 (maximum 4Mb of virtual memory, 10-15Mb for the user on disk, CMS, EXEC2, XEDIT and DGF available), so as to implement the complete system on a micro, with all its subenvironments.

The organization of the memory during execution is as follows.



As a matter of fact, the generalized supervisor is itself an interpreter for the ILSD. This generalizes the idea

underlying the implementation of all components of the system (the SLLPs).

3.3. Uniform generation of software and lingware "modules"

This trend toward uniformization is also visible in the uniform treatment of software and lingware modules. From the data-base point of view, the main difference is that the software modules are kept on a share system space, whereas the lingware modules are stored in the user space, and are created or erased under the user's control.

From the internal point of view, there is a necessary difference of structure of these modules, because the compilers of the SLLPS don't produce executable machine code, but a code for some abstract machine simulated by the "interpreters" of the SLLPs. However, this difference is not visible at the external level.

CONCLUSION

Although there is a parallel effort to develop an entirely new (LISP based) CAT system, in the framework of our national project, we hope that the effort invested in ARIANE-85 will prove to be fruitful. First, the development of various ARIANE-78-based MT systems has been scheduled in such a way that it will be possible to make use very quickly of the new possibilities offered by ARIANE-85.

Second, the fact that the underlying software is largely written in low-level programming langages has certainly slowed down the implementation. But, in the current world of microcomputers, it might very well prove important that this very sophisticated system can run on new IBM micros, with a price tag less than 10% of that of a LISP machine, with comparable projected performance.

Third, it may be interesting to note the evolution of the specialized data-base monitor toward a complete programming environment, organized around a set of specialized languages for linguistic programming and for the description of sequences of phases or subphases.

REFERENCES

1 . J. Chauché (1974),
"Transducteurs et arborescences. Etude et réalisation de systèmes appliqués aux grammaires transformationnelles",
Thèse d'Etat, Grenoble, décembre 1974.

2. J. Chauché (1975),
"Les langages ATEF et CETA",
AJCL. Microfiche 17, 21-39, 1975.

3. Ch.Boitet (1976),
"Un essai de réponse à quelques questions théoriques et pratiques liées à la Traduction Automatique. Définition d'un système prototype",
Thèse d'Etat, Grenoble, avril 1976.

4. Ch.Boitet, P.Guillaume, M.Quézel-Ambrunaz (1978),
"Manipulation d'arborescences et parallélisme: le système ROBRA",
Proc. of COLING-78, Bergen.

5. B.Vauquois (1979).
"Aspects of automatic translation in 1979",
IBM-Japan, Scientific Program, July 1979.

6. Ch.Boitet & N.Nedobejkine (1981),
"Recent developments in Russian-French Machine Translation at Grenoble",
Linguistics 19, 199-271 (1981).

7. A.Melby,
"Multi-level translation aids in a distributed system",
Proceedings COLING82, North-Holland, 215-220, Prague, July 82.

8. Ch.Boitet, P.Guillaume, M.Quézel-Ambrunaz (1982),
"ARIANE-78: an integrated environment for automated translation and human revision",
Proceedings COLING82, North-Holland, Linguistic Series No 47, 19-27, Prague, July 82.

9. Ch.Boitet & N.Nedobejkine (1983),
"Illustration sur le développement d'un atelier de traduction automatisée".
Colloque "l'informatique au service de la linguistique", Université de METZ, France, juin 1983.

10. B.Vauquois (1983),
"Automatic Translation",
Proc. of the summer school The Computer and the Arabic Language, Ch. 9, Rabat, October 1983.

11. R.Gerber (1984),
"Etude des possibilités de coopération entre un système fondé sur des techniques de compréhension implicite (système logico-syntaxique) et un système fondé sur des techniques de compréhension explicite (système expert",
Thèse de 3è cycle, Grenoble, janvier 1984.

12. Ch.Boitet (1984),
"Research and development on MT and related techniques at Grenoble University (GETA)",
Lugano Tutorial on Machine Translation, April 1984.

13. J.Slocum (1984),
"METAL: The LRC Machine Translation System",
Lugano Tutorial on Machine Translation, April 1984.

14. Ch.Boitet & R.Gerber (1984),
"Expert Systems and other new techniques in MT",
Proc. of COLING-84, ACL, 468-471, Stanford, July 2-6, 1984.

15. D.Bachut & N.Verastegui (1984),
"Software tools for the environment of a computer-aided translation system",
Proc. of COLING-84, ACL, 330-334, Stanford, July 2-6, 1984.

16. W.Bennett & J.Slocum,
"METAL: The LRC Machine Translation System",
Linguistic Research Center, Austin, Texas, USA, September 1984.

-o-o-o-o-o-o-o-