[*Proceedings of a Workshop on Machine Translation*, July 1990, UMIST]

METAL: Computer Integrated Translation

Gr. Thurmair
Siemens Nixdorf AG

Abstract

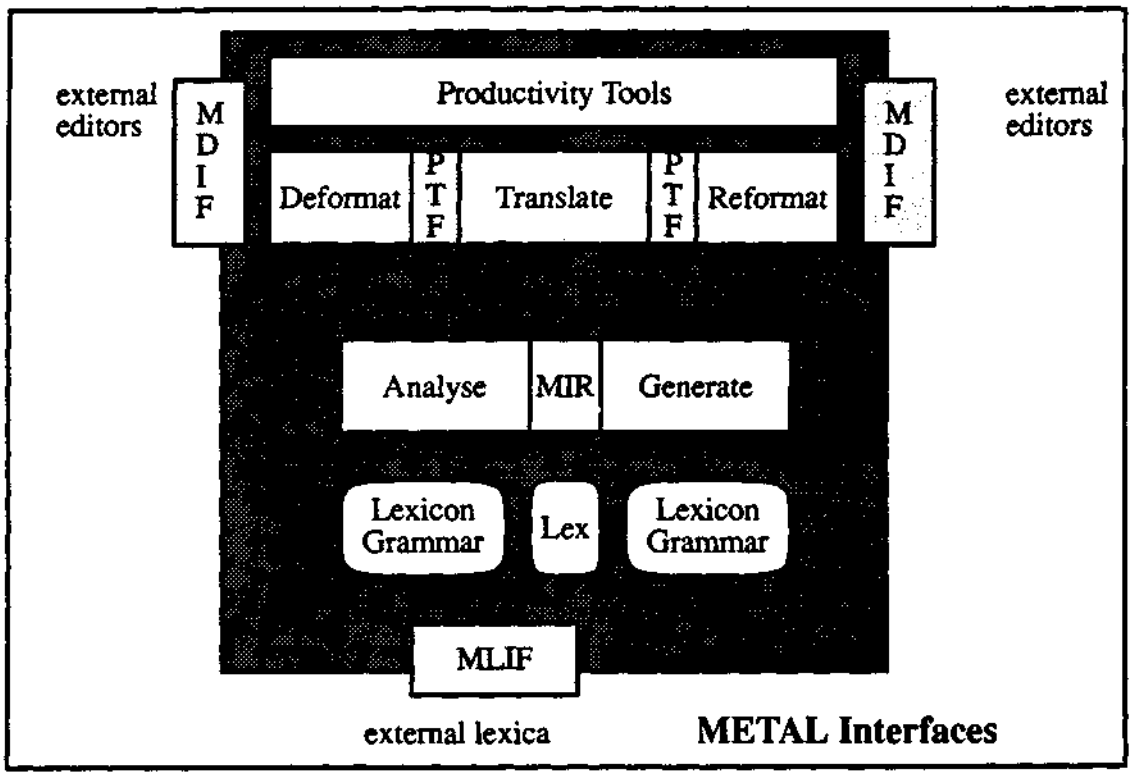
The following paper describes the METAL machine translation system in its present state, as part of a computer integrated translation environment. During the past few years, the system has been considerably refined and extended, although the design of some main parts has remained unchanged. These changes and extensions are due to the fact that METAL has become an operational system; this is explained in the first chapter. The second chapter describes the architecture of the text handling environment which has become more and more important; the third chapter deals with the translation core engine. The remaining chapters deal with productivity and maintenance tools.

1. Changes in the METAL architecture

Before METAL could enter the market in 1988, the system had to be changed considerably compared to the earlier prototypes: We had to reprogram considerable parts of the core system; we had to broaden the coverage of the grammar as well as the lexica; we had to add completely new components for text import / export and text handling; also a special purpose editor had to be developed for quick postediting.

It turned out that the possibility of embedding a MT system into the document processing environment is of crucial importance for the success of such a system. Users must be able to integrate the system into their own documentation environment, both from a hardware and software point of view, but also with respect to the specific terminology they want to maintain. Therefore METAL had to change into an open system, which means providing interfaces to the outside, and between the main system components.

- first, we developed an interface for **document interchange,** called "METAL Document Interchange Format (MDIF)". This enabled us to convert different external document formats into METAL structures, without influencing the system kernel.
- the deformatting tool processes pure text into what we call **Plain text Format** (PTF). In this format, the text is sent to the machine translation tool as well as to other linguistic processing tools (like Terb bank lookup components).
- second, we developed an interface to import and **exchange lexical data,** called "METAL Lexicon Interchange Format (MLIF)". This turned out to be necessary as METAL customers wanted to import their existing lexica and terminology into the METAL system.
- finally, we are developing an interface structure to produce **linguistic structures** called "METAL Interface Representation (MIR)". This is because the system had to add new language pairs and become more interlingual.



As a result of these changes, the system architecture became well structured, which enabled us to work in a much more modular environment. The system architecture now looks as follows:

In addition to these design changes, enhancements and extensions of the single components have been earned out to improve the overall performance of the system: It turned out that the success of a MT system depends on its ability to improve the overall performance of the task to produce multilingual documentation; and the actual translation is just a subpart of this task.

2. The Text Handling Environment

This component runs on a standard UNIX machine and is written in C. Its task is to convert and deformat documents so that they can be processed by the core translation machine, to reconvert the translated documents, and to aid the users in postediting and revising their documents.

2.1 The METAL Document Interchange Format (MDIF)

This format has been created to allow for easy connection of “foreign” document formats to METAL. It is a kind of markup language which marks special document properties which are relevant for translation, like fonts, tabulators, paragraph properties, headers and footers, frames, anchors, etc. The respective items are marked in a special and uniform way. The text itself is converted into ISO 8859/1 code; characters which are not representable undergo special markings.

MDIF has been created as a special purpose markup language, designed on the basis of experiences with standard text processing and publishing systems. It takes only those items into account which deserve special attention in text processing: Fonts must be preserved, anchors must not be deleted, etc. Other information, like graphics, bitmaps etc., are not relevant for translation purposes and are not part of the interface.

During further processing, the items marked can be easily accessed and interpreted by a special parser designed to operate on MDIF structures. This shows the advantage of the MDIF kind of interface: The inner system parts are not influenced when a new "foreign" document format is added.

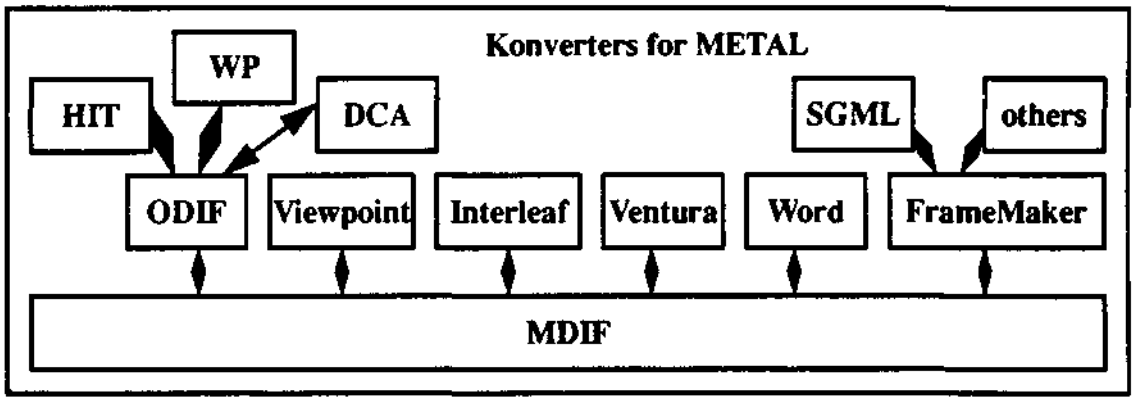
As MDIF is basically an ASCii file, “Human Readable MDIF” files can even be used for human translation purposes, e.g. for external translation offices.

2.2 Converters

In order to support the MDIF format, external document converters have to be written. These converters consist of two parts: First, the foreign document format is parsed (e.g. Word or Viewpoint Document structures must be analysed), and then, the recognised items must be converted into MDIF format. When reconverting MDIF into external document formats, the reverse direction has to be taken.

As it is a time consuming task to build converters, and as there are a lot of text processing systems, the strategy is to concentrate on available standards. Such standards exist for the “lower level” text processing systems, developed in the context of the “Office Document Architecture” and the “Office Document Interchange Format”. METAL supports the existing standards (Q111 and Q112), to which a variety of conventional text processing systems can be added, like WordPerfect, DCA, or HIT (cf. Garcia 1991). In the area of high quality publishing systems, however, good standards do not exist; therefore METAL offers converters for Interleaf, Xerox-Viewpoint, FrameMaker (which in turn has its own Filterpak system, also supporting SGML), Ventura, and also more sophisticated Word applications.

Parts which are not relevant for MDIF, like graphics, some general document layout information etc., are led around METAL and merged with the translated text during reconversion.



All METAL relevant document parts are recognised by the converters and marked according to the MDIF notations. The following system components can work independent of the origin of the document to be processed.

2.3 Deformatting and Reformatting

MDIF documents have to be brought into a form in which the translation core engine can process them; i.e. the documents have to be converted from a MDIF format into a "Plain text" format (PTF). This format consists of the sentences of the text, and some interspersed parameter lines.

Deformatting is performed by a sequence of single steps, like table (or multi-column layout) recognition, marking of abbreviations, sentence boundary recognition, font and anchor treatment, etc. It can be controlled by the users via a special parameter file which contains all parameters necessary for the translation. The different modules run in batch mode and accumulate markups according to their different tasks.

The resulting file provides another interface, well suited for both machine translation input and other applications. As it consists of the "pure text", without any non-linguistic information occurring in documents, it can in particular be used for different linguistic purposes, as it contains only linguistic information: Concordances, hypertext applications, terminological applications etc. are supported by this format. In the METAL context, PTF files are transferred to the LISP process of the core engine for translation. METAL has the possibility of job maintenance: Different priorities, several scheduling and monitoring functions are offered to make the system more flexible.

Reformatting takes the machine translation output file, again in PTF format, and produces the original text configuration by rearranging the text blocks, distributing fonts, anchors, etc., and removing superfluous markups.

2.4 User Tools

In order to support users in the translation tasks, a set of tools has been developed which increases the efficiency of their work.

METAL offers a special **editor** for postediting. It has been designed to support corrections of machine translated texts and to minimise the effort to overcome frequent system errors, like cutting and pasting sequences of words, moving phrases, changing capitalisation etc. All these functions can be done by the touch of just one key. Users can work on an interlinear version of the text, or in a two window mode. The editor is based on standard UNIX technology.

In order to prepare the text for translation, or to speed up postediting, METAL offers the **"METAL Pattern Matcher".** This tool allows users to define patterns and actions on those patterns. The pattern Matcher is useful before as well as after translation: E.g. before translation, some text parts can be excluded from translation (e.g. acronyms), others can be rephrased (e.g. "5cm" → "5 cm"), etc. In the postediting phase, system errors can be corrected.

The Pattern Matcher accepts regular expressions, from simple strings to complicated expressions with wildcards etc.; it also offers some predefined string classes, like numbers, letters etc. which users can use. Moreover, it supports document formating information: Fonts, anchors etc. do not get lost if a pattern is changed.

Another useful tool is a **text comparison** facility. Often, only parts of texts have to be translated (e.g. if a second version of a software product is developed). In this case, the system should be able to reuse the document parts which already have been translated (and postedited). Only the new text parts have to be found and translated.

The document comparison tool operates on machine translation input and output files and produces a delta input file. This file is translated, postedited, and merged back into the original version. Internally, it uses a fuzzy matching algorithm which allows for extensions into interactive translation based on databases with predefined texts. At present, however, only perfect matches are considered.

3. The core translation component

The core functionality of the system is the translation engine; this is a special LISP process. It consists of a language independent software kernel, of language-specific lexical and grammatical knowledge-bases, and of development and productivity tools.

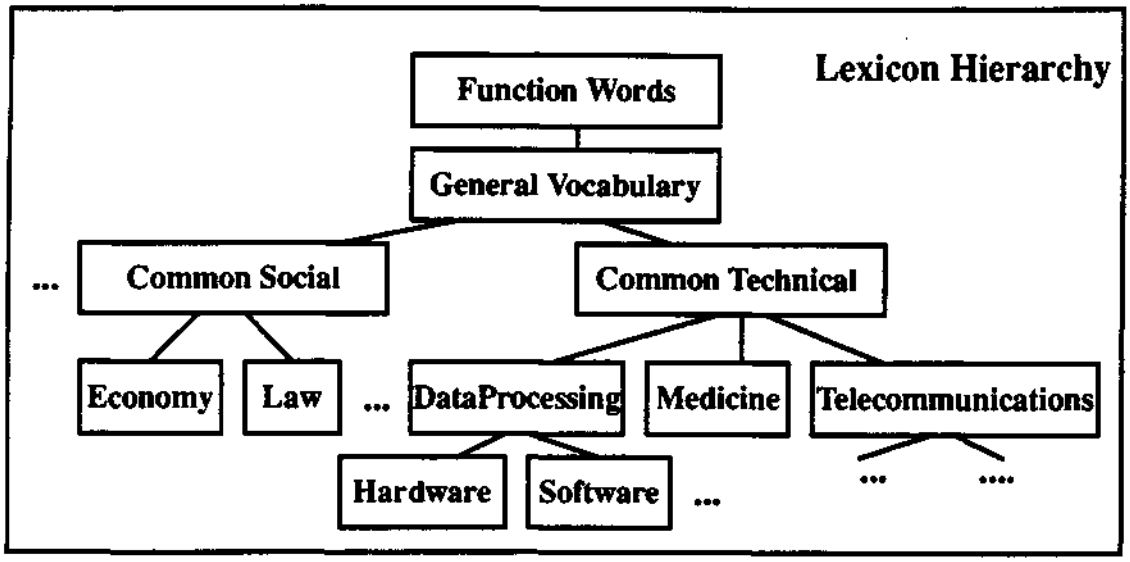
3.1 The METAL lexica

The METAL lexica are organised according to the principle of modularity in two respects.

First, the METAL lexicon database is organised according to the different languages, i.e. it contains mono- and bilingual lexica. Every language pair uses a source language monolingual, a bilingual transfer, and a target language monolingual lexicon. This allows for re-use of parts of the lexicon databases in new language pair configurations. However, there will always exist entries which are used only for analysis or for generation. These entries are marked accordingly. Transfer lexica are unidirectional; experiments with bilingual lexica have show that they are both theoretically inadequate and practically not maintainable.

Second, the lexica are organised according to sublanguage modules in a tree-like manner. The top level consists of function and general vocabulary entries, followed by two modules covering common technical and common social vocabulary. Below those, there may be different submodules for topics like economy, data processing, medical engineering, etc. The hierarchy can be configurated by users according to their requirements. This organisation enables the interchange of lexicon modules as well as the specialisation of the lexicon database for particular users' needs.

For a translation run, users can choose which lexicon modules in which order should be used. This enables the system to offer the most specific translations before the less specific ones.



3.1.1 Monolingual Lexica

The monolingual lexica contain between 20000 and 90000 entries currently, depending on the language, and on the requirements of a user. They exist for English, German, French, Spanish, Dutch, and Danish.

The monolingual lexica are collections of features and values; they basically contain morphosyntactic information, like syntactic category, allomorph, morphological class, gender, abbreviation/acronym, argument structures, position information, etc. Moreover it contains semantic information in terms of feature semantics. METAL prefers to have less features which can be consistently coded, than to have many unreliable features. Finally, it contains maintenance information, like name of the author, coding date, status, coding site.

The values of the features need not be atomic; they can also be lists or sets. METAL uses specification files which declare for each category which features are obligatory and optional and which values of what type are used. These declaration files are also used for data compaction in the lexicon database.

3.1.2 Bilingual Lexica

The bilingual lexica are unidirectional. We abandoned the approach of having bidirectional transfer lexica, as it is not particularly useful in cases of 1:many transfers (they depend on the translation direction), and it has severe drawbacks in maintenance (e.g. in case on deletions in one direction: Does this imply deleting the reverse as well?). Therefore the METAL transfer lexica are unidirectional.

Every entry consists at least of source language category and canonical form, target language category and canonical form, and a preference.

In addition, in cases of complex lexical transfers, there are specifications of the structural environment which must hold for a given transfer, and also a specification of the target structural environment which this particular entry produces. These specifications are

(feature and structure) tests, or basic adding, mapping and deleting operations (cf. Thurmair 1990):

es ist mir egal → I do not care about it

SL-CAN egal + SL-CAT ADJ → TL-CAN care + TL-CAT VB

Tests: existence of a DATIVE-NP

Operations: MAP: DATIVE-NP into SUBJECT-NP
MAP: SUBJECT-NP into ABOUT-PP
REVERT-NEGATION

These operations are not immediately executed during transfer, but the nodes in question are marked with appropriate features, and the actions are carried out during generation.

As in the case of monolingual entries, maintenance information is given in transfers as well (author, coding site, date, comments). This is needed for maintenance and copyright purposes.

In the case of l:many transfers, the problem of which transfer should be tried first has to be solved. In particular, sublanguage requirements and linguistic tests interact here: If a user specifies a test on a transitive verb in the data processing area, what should be taken if the system finds a transitive verb, but does not translate in the medical engineering area: The (non-matching) transitive entry (which leads to wrong sublanguage specifications), or the (default) intransitive entry (which leads to linguistically wrong decisions)? Former METAL versions allowed the users to specify the order of tests themselves by using preferences; but this created problems in importing and exporting lexicon modules, as the relative order of newly imported entries had to be fixed by hand. Therefore we developed a special scheme which computes the preferences automatically using “soft” and “hard” tests and allows for easy sublanguage lexicon import

3.2 Grammars

METAL uses an analysis and a generation grammar for each language pair, generation is a subset of the analysis grammar and uses the same formalism. Bidirectionality is a questionable goal for large coverage grammars as they always must analyse much more than they would want to generate; in particular in languages with relatively free word order, analysis is much more computationally expensive than generation. Also, robust grammars must cover some ungrammatical and ill-formed, however frequently used, input structures, like punctuation errors; but we would not want to generate these structures. Therefore it seems to be more effective to have two grammars for analysis and for generation, using the same formalism.

3.2.1 Morphology

Morphology is done in two steps: Morphological decomposition and morphological analysis. The latter uses the standard grammar rules and grammar formalism as described below.

Morphological decomposition has to compare the input strings with the lexical entries. This is done by ordering the lexical database twofold: There is a tree of the letters of a word (or an optimised hashing version of it), and if a node is a legal word it delivers a pointer to the real entry in the database.

Every incoming word is compared with the letter tree, and all possible combinations of morphemes are delivered, together with some additional information about placement, orthography, font, etc. If a word cannot be analysed it is marked as unknown.

As large lexica with many short words (nearly every letter can be a word of its own) produce dozens of possible decompositions, METAL restricts these possibilities using a transition matrix which controls the possible combinations of morpheme class transitions (like: “an inflection must not follow a prefix”, “a noun can be followed by a verb inflection only under special conditions”). This transition matrix does an initial filtering of unwanted decomposition results.

Usually, there are still several possible combinations of morpheme classes. They are given to the grammar for further disambiguation and analysis. The grammar has special rules for morphology; inflections, derivations, and compounding are treated.

The output of the morphological decomposition component is a chart containing possible morpheme sequences; each morpheme consists of its lexical information, enriched by some system features, like orthography, font, word placement, category (e.g. number, unknown), and others. Also, each morpheme has a score specified in the lexicon; it makes the distinction different lexical readings, multiword entries and others; it forms the starting point of the preferencing and scoring mechanism.

3.2.2 Grammar Formalism

METAL uses a formalism which combines declarative and procedural elements in an effectively running system. Unlike ATNs which are completely procedural, the grammar consists of rules which operate on augmented phrase structures. Unlike completely declarative grammars, METAL uses procedural elements to speed up processing efficiency.

The basic translation strategy is to transform trees into other trees: The source language parse tree is transformed into a (normalised) interface tree, and this tree generates the target language surface structure tree. To do this, METAL uses transformations which describe the tree layout and its feature decorations, and specify the target tree and feature structures (wildcards for nodes are supported). Compared to other frameworks (e.g. Eurotra’s t-rules), METAL does not apply these transformations in a separate step as this constitutes technical overhead, but in the analysis rules themselves, instead.

The METAL grammars consist of rules. A rule is based on a phrase structure X-bar categorial backbone; it consists of several parts:

- a context-free phrase structure part
- a section where feature and structure tests are performed
- a section where the top node is described, equipped with features, and the local tree is transformed into normalised structures
- a type information (morphological, syntactic)

- a level information for the static weight of a rule
- several sections with maintenance information, like a unique rule-id, author, examples, comments

Throughout the grammar, METAL uses a grammar language based on operators; this is a meta-language which is compiled into LISP code later on. Such operators are "required", "not required", "intersection" and others for tests; "from", "to", "put" and others for feature percolation, "transform" for structure transformations. Operations used in several rules can be put together into special procedures (similar to templates in other formalisms) and called if appropriate; this facilitates grammar maintenance.

Although it is less declarative, such an approach is more powerful and more efficient than existing unification-based approaches; and it enables the concentration on language phenomena to occur without being restricted by formalism discussions.

3.2.3 Grammar coverage

The coverage of the METAL grammars is different depending on the languages to be analysed; in general, it is oriented towards technical documentation, like computer manuals, chemical texts, or banking abstracts. However, there is no special tuning of the grammar for certain sublanguages (This could easily be done by adjusting the rule levels to the different text types, based on statistical corpus analyses).

As far as the grammar is concerned, sublanguages differ in the preference they make of syntactic constructions rather than in their use at all; with very few exceptions, all grammatical constructions may occur in all sublanguages but with different frequency; therefore it seems to be better to regroup existing grammars according to different rule frequencies than to write different grammars for sublanguages.

The METAL grammars follow the X-bar schema in their organisation and coverage. Beyond the standard structures of NPs, PP, APs, adverbials, and complex predicates, some of them cover morphological phenomena as well, like composition (German) or derivation (English). They also cover phenomena like coordination (also with different heads), the most frequent gapping phenomena, control and raising issues, modals, tense and aspect, definiteness of NPs, preposition stranding phenomena in English, extraposition problems in Dutch, and many others. The goal is to cover most of the structures which can be found in technical texts that are not too complicated; they sometimes differ considerably from the "hot issues" in linguistics; e.g. they include ungrammatical structures, which are tried to be covered by fallback rules.

There are two components of special interest in the system. The first one is verb framing. Verb framing, checking for completeness and coherence, is done if a sentential node is built. At this level, the system matches the frames as specified in the arguments section of the lexical entries against the structures found in the sentence. This operation can become highly complex in cases of optional constituents, of complex role descriptions (nominal or sentential role fillers of certain types), or of several candidates for role fillers the combinatorics of which have to be calculated.

This is supported by software which allows linguists to specify what they want the system to look for; the combinatorics and the matching are done by the system software for efficiency reasons (cf. Gebruers 1987).

The second component with particular software support is anaphor resolution. This is done at the end of the analysis, using the notion of C-commanding and following an improved version of Hobbs' algorithm. Potential antecedents are identified based on syntactic and semantic properties.

3.2.4 Control

Large grammars need special treatment of complexity problems, as the interaction of many rules can cause the system to explode combinatorially. Therefore special care has to be taken towards efficiency; this is done by grammatical as well as software means.

The grammar orders its rule according to their probability to contribute to a successful parse: The more often they are used the higher their level is. This can be measured statistically. As a result, METAL has rule levels where the standard phenomena are covered first, the exceptions and ungrammatical structures later on: The system should apply lower level rules only if there are no standard rules which lead to a successful parse.

Based on this schema, the software has to use effective parsing strategies.

After testing the available parsing strategies, the parser with the best practical performance has been adopted (Slocum 1981). METAL uses an active chart parser which works middle out, left comer, some paths. It uses an agenda divided into a hundred levels and fires all the rules belonging to an agenda class. If among those a complete spanning interpretation can be found, the parser stops; else it rearranges the agenda and fires the next best rule set.

This means that METAL in general delivers only the "best" reading of an input sentence; it does not collect all possible ambiguities or offer several possible translations to the users; this would take too much time in postediting. Therefore, the most likely interpretations must come out of the analysis first.

Rule application is driven by a preferencing and scoring mechanism (Caeyers 1990). Every hypothesis obtains a score, originating in the lexicon (e.g.: higher scores for multiwords). The score of a hypothesis is composed of the level of the rule which built it, together with the scores of the sons of the partial tree considered. But grammar writers have also the possibility to influence the scores by boosting or lowering certain constructions in the grammar rules. Apart from this last property, the scoring mechanism reliably finds the best scoring hypotheses first.

This properties of METAL has the effect that usually, the intended interpretations of a standard sentence come out first, and that combinatorial explosions are reduced to a minimum.

3.2.5 The METAL Interface Representation

The output of the grammar is a structure which follows the METAL Interface Representation (MIR). When METAL had to become a multilingual system, it turned out that a pure transfer approach was suboptimal: As in a pure transfer approach every source language structure is mapped into a corresponding target structure, as many target structures have to be built as there are source language components. Therefore, METAL uses a **transfer interstructure** (Alonso 1990) approach now: We build a common representati-

on which all languages support (as far as possible), and from which all generation components can start; the difference to an interlingua is that these structures have lexical entries at their terminal leaves, and consequently use transfer lexica.

The METAL Interface Representation is similar to the Eurotra Interface Structure: It specifies the tree layout in terms of dominance and preference relations, and it specifies the features and values expected at the different nodes. Compared to the Eurotra Interface Structure, it is more cautious in terms of node pruning (prepositions, modals), and more liberal in terms of canonical orderings.

In order not to be too rigid and to force languages to counter- intuitive decisions, there is a certain degree of freedom in the single language components. MIR structures therefore can differ in some minor respects in the different languages. It is a matter of structural transfer to map these differences into the target MIR representations.

3.3 Transfer

33.1 Transfer Organisation

The task of the METAL transfer is to produce correct input structures for generation. To achieve this goal, the system works top down the MIR tree recursively. The transfer can be controlled by the linguist, and different strategies can be followed, e.g. transfer of heads before transfer of their modifiers (Alonso 1988). There are two main tasks to be performed:

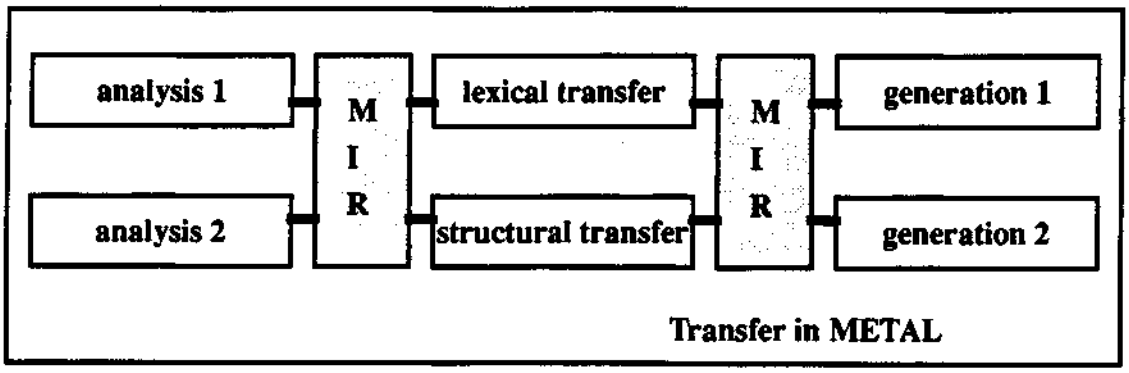
First, **structural transfer** has to be performed. It performs some default changes of idiosyncratic language phenomena, and it puts some generation instructions (in terms of features) onto the nodes in question (e.g. “build an English relative clause from a German complex adjectival phrase”, or “build a Spanish PP from a German compound specifier”).

As it turned out that complete identity of all MIR structures of all METAL languages is not feasible, we have some adaptation components (called “source-to-MIR”) for the specific transfer to be performed.

Second, **lexical transfer** has to be done. Again, this can lead to additional instructions for the generation component; this happens in cases of complex lexical transfers, as described above (Thurmair 1990): Lexical transfer triggers changes in the structural environment of the lexemes.

From a logical point of view, METAL tries to transfer structures into structures, not words into words. In the case of verbs, for instance, the basic transfer unit is the verb frame: We want to map source verb frames into target verb frames; this implies not just replacing the canonical form of the verb but also manipulation of roles. But it has to be taken into account that also non-role (or non-argument) constituents undergo structural transfer.

In order to be able to do this, lexical transfer must not take place at the terminal level of the tree, as only local information is available there. Instead, transfer of the head categories is done at their maximal projection levels, where all the modifiers of the head are accessible for transfer operations. The basic control structure therefore is a mixture of



contextual transfer of the heads of constituents on XP level, and of local transfer of modifiers at the terminal levels.

3.3.2 Multiword transfer

This concept also enables METAL to deal with multiword transfer. As long as they are not fixed syntactic structures like support verbs, they are treated just as any other syntactic component. The fact that they are special semantic units does not influence their syntactic behaviour (otherwise we would obtain two readings for each of these concepts, the normal one and the idiomatic one).

If there should be a special transfer for these constructs, this can be stated in transfer lexicon tests and operations, e.g.

SL-CAN **Steuerung** + SL-CAT N → TL-CAN **timing** + SL-CAT N

Tests: SL-Adjective **zeitlich**
Operations: TL-delete **zeitlich**

Before the actual generation can start, the transfer changes must be evaluated, and the respective tree changing operations are triggered.

3.4 Generation

Generation expects a tree in the MIR format of the language to be generated (There are procedures to test MIR compatibility on incoming structures). It results in a tree with the proper allomorph at the terminal nodes. Afterwards, we collect the allomorphs, redistribute the font and anchor information and write a PTF record into the output file.

Generation itself uses a special generation grammar which is triggered by the category of the node to be generated. The grammar consists of sets of generation rules which are fired in sequential order. They use tree transformations to create the appropriate target structures.

The METAL grammars are not bidirectional: For large grammars, the analysis coverage will always be larger than the generation necessities. In German word order, for example, much more variations will have to be analysed than we want to generate. Considering

this, it is much more effective to do generation in a straightforward way, controlled by certain features (like topicalisation).

Lexical generation is performed in three steps. In transfer, we consult just the transfer lexicon where the correct lexical unit is found. In generation, we consult the target mono lexicon to fetch information about the target mono entry, e.g. its allomorph, determination behaviour, etc. Finally, we inflect this lexical unit. This can be an iterative process (e.g. if a verb participle is used adverbially). Allomorph selection and inflection are controlled by special tables which states which feature combinations are relevant for the respective step.

4. METAL Productivity Tools

We have to take into account that a MT system is always embedded in processes that also include document production and human translation. METAL therefore must be considered not just as a MT "blackbox" but as a Computer Integrated Translation (CIT) tool which supports tasks which need to be performed in the translation context, whether or not translation is done by humans or by machine. Some of them are described in this section.

Experience shows that MT systems must be seen as just one of the tools in a CIT enviroment; there are many other tools which support both human and machine translations; there are tools just for humans (like terminology lookup), and there are machine specific topics. The requirement is that all those tools together form a productivity environment, a "Translator's Workbench", of which MT is a part. This has consequences for the MT system itself, e.g. the lexica must support both human (i.e. terminological) and machine information.

4.1 Converters

Based on the converters which have been written to be able to process the most widespread editors and DTP systems, METAL has developed a possibility to aid human translators with converters: They should take advantage from the MDIF format by also being able to use the MDIF interface for human translations. The advantage is that they can use their standard wordprocessors and translate texts produced by sophisticated DTP systems, without losing the layout information of these systems. This increases human translation by saving re-layouting time.

4.2 Pattern Matcher

In METAL, we developed a special pattern matcher for search and replace operations. It is as powerful as a regular grammar in describing expressions to be changed. The patterns, consisting of search and replace expressions, are stored in files which are hierarchically ordered (text specific, user specific, system specific, etc. They can be called both in the pre-editing and the post-editing phase.

It is used in METAL to identify dates, constants and acronyms, filenames etc., things which should not be translated. It can also be used to replace very standardised source language expressions by the respective target language expressions. Finally, it allows

posteditors to correct system errors in a general way ("does not be → is not"). The pattern matcher is a powerful tool in the METAL environment because it improves the MT text input quality, and because it speeds up the postediting time,

Again, this tool could also be used for human translation, by pre- translating highly repetitive texts, this possibility is currently undergoing internal tests.

4.3 Document Verification

One of the advantages of a MT system is that terminology, once stored, can be used for later versions of a text; therefore the effort of building up lexica decreases, however, this advantage of MT can only become valid if it can cope with changes in translation input texts: If a new version of a text is to be translated, its terminology will have already been coded. But if the system is not able to identify changes in a text, the whole text must be re-translated and re-postedited. This is not acceptable as soon as postediting has to be done: Users want to postedit only the text parts which actually have changed.

Therefore METAL needs a tool which enables it to compare different versions of a text, merge the target language parts of the text parts which have remained unchanged with the new source language parts, and produce a bilingual text for MT input. Only the new source language texts must be translated and postedited.

Many of the existing text comparison programs are either not powerful enough (e.g. they do not identify identical text portions placed at different text parts), or they are tied to a special editor or DTP system. Moreover, the connection of source and target text portions is a tricky task, as the target portions can have been postedited. Therefore, METAL developed a tool, using the converters and the MDIF interface, which compares text portions with previous versions based on MDIF paragraphs.

Again, this tool is useful for human translators as well; in larger translation departments, translations of the second or third versions of a text nearly outnumber primary translations.

5. METAL development tools

In order to develop the system, METAL has produced a powerful set of tools, in particular for lexicon and grammar work.

5.1 Lexicon Maintenance

The METAL lexica are accessed by a set of user programs and commands. Most of them are designed for end users, some are only for experts.

5.1.1 Coding of entries

Experts have a set of commands, like "edit source language entry", "print transfer entry" etc.; the respective entries are loaded into the editor, printed, deleted etc., in the system internal format (i.e. as lists of features and values). Also, look-alike entry specification is possible ("edit 'floppy disk', it 'goes like' 'disc'"). For experienced coders, this is the fastest way of coding; however, they must know the internal values of the lexicon.

Example of a monolingual lexicon entry (internal format)

CAN	improvement	canonical form
		morphological information:
ALO	improvement	alomorph
ON	VO	onset: vocalic
CL	(P-S S-01)	morphological class
		syntactic information:
CAT	NST	category: noun stem
DR	(NP RD)	determiner requirement
KN	MASS	kind of noun: mass
FC	PP	form of complement
MC	"in"	marker of complement
		semantic information:
TYN	ABS	type of noun: abstract
SX	N	natural gender
		maintenance information
Author	Boone	author
Date	5-Aug-91	date
Site	LRC	site

Users need not know this. They are given a interactive coding tool, called Intercoder. This is a window system for entering and modifying entries.

The Intercoder consists of a language independent software kernel and language specific tables which define the respective window system for the different languages. The user surface consists of windows for specifying entries, for editing them, for messages and examples, and for storing the coding history. The history window can be used for actions specific to a coding session (like consistency checks).

5.1.2 Defaulting entries

If the users specify entries which are not in the database yet, a defaulting component is called which tries to default the values for the features for the respective entry. This is done based on heuristics about the incoming string (affixes, compound parts, etc.). The defaulters can be improved to set up rapid tools for mass coding.

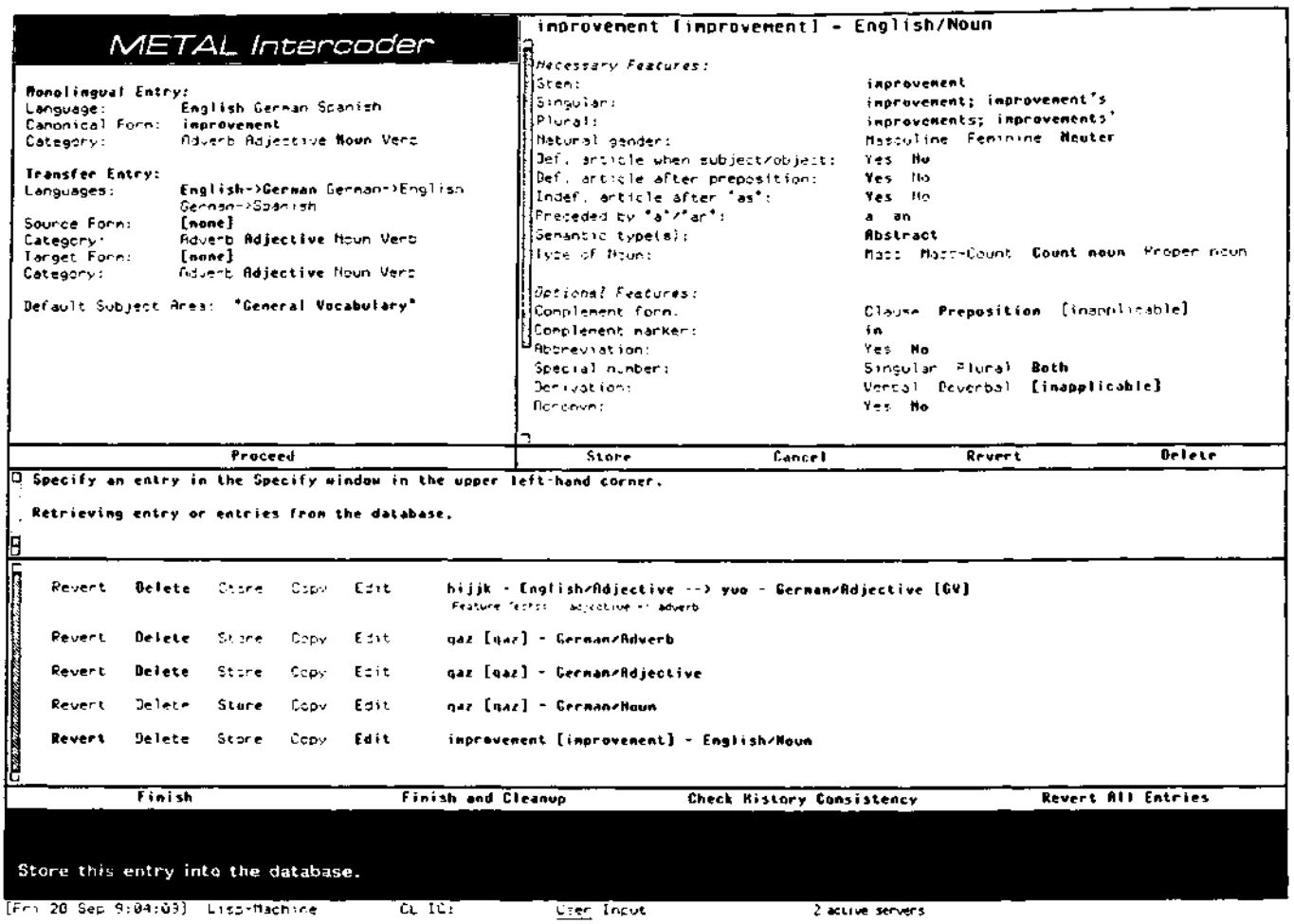
The defaulter is also used at runtime, to be more precise if an unknown word is encountered. Online defaulting is described in Adriaens 1990.

5.1.3 Lexicon querying

Users have special options to query the lexicon. Queries consist in boolean expressions for any combination of features and values. The result of the query is a set of records which can be edited, printed, displayed, etc.

5.1.4 Lexicon Interchange

METAL has the possibility of exchanging lexicon modules, importing and exporting them, and merge them into existing lexica. This is needed as METAL offers lexica for



Intercoder: coding surface for "improvement"

special purposes to its customers, and as customers want to exchange their lexica as well.

To be able to do so, an interface (called MLIF- Metal Lexicon Interchange Format) has been created which supports the exchange of METAL lexicon modules, but also some external formats (needed to convert existing lexica of new customers into the METAL format).

Conflicting entries are dealt with in a special window for merging. This window shows the conflicting entries in different windows and allows the users to choose the entry they want to keep (or edit all of them to create a new entry).

5.1.5 Lexicon maintenance

METAL offers the possibility to run tests on the lexical database to keep them consistent. Users can compare source, transfer, and target entries to find gaps in the lexica; they can search for doublets in the lexica; they can test transfer tests against monolingual entries (e.g.: test on a verb argument in transfer which was not specified in the mono); they can test for syntactic wellformedness of entries, etc.

5.1.6 Version management

As lexica in METAL are coded by different people at different sites, care must be taken to manage different versions. Coders start coding in private files; those files are collected

and merged in order to create new lexicon releases. This way, there is always an official and tested status of a lexicon release.

The same technique is also applied to syntax development: Grammarians work on their private copies of the system and create their own working environment which they can load and modify; for new releases, all changes are merged into the new official version of the grammar. This is again supported by software (finding deltas in the rules etc.)

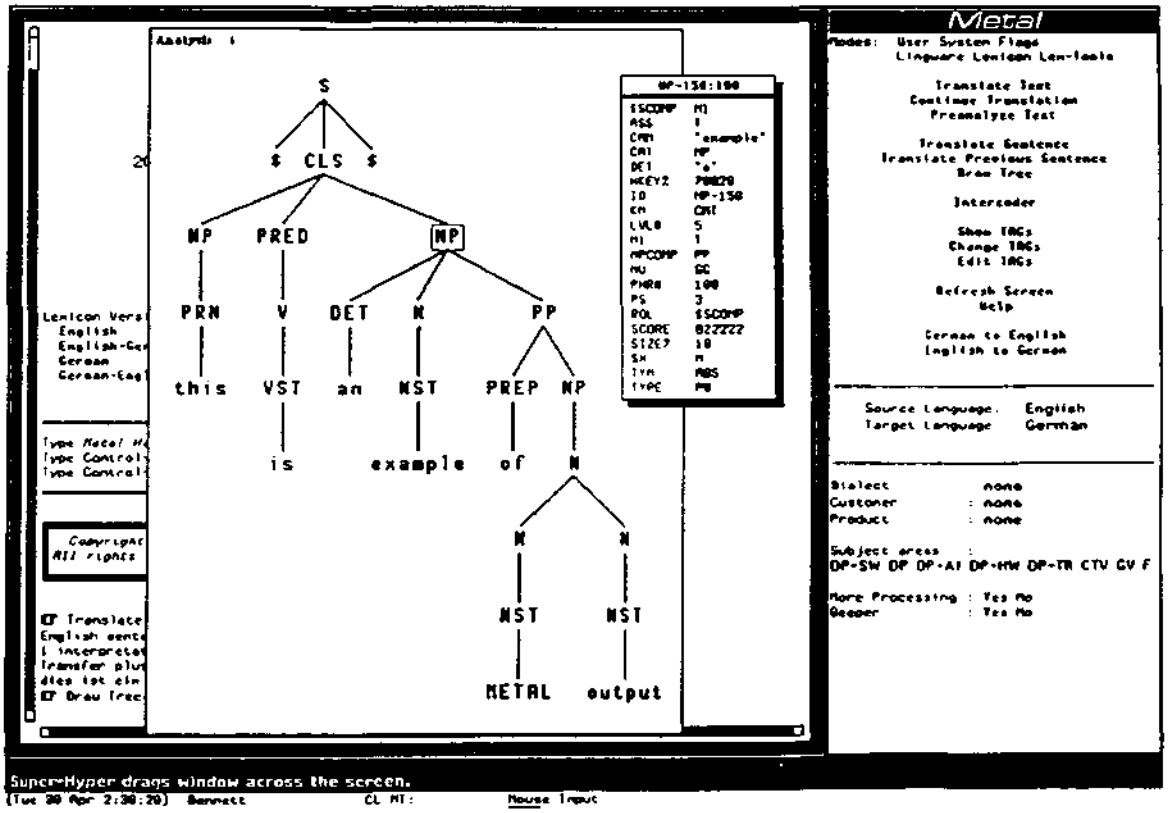
5.2 METALSHOP Syntax development environment.

In order to work productively, grammarians need powerful tools for syntax development. Therefore, the syntax development environment METALSHOP was created. It supports the following activities:

Draw Tree:
At all levels, the grammarians have the possibility of drawing the current tree; they also can click on all nodes to see the feature decoration of each node, and they can see the "feature history" (Where has a certain feature been set or modified?) as well as the feature documentation.

Draw Transformations:
This is a possibility to see the current tree, the structural description of a transformation to be applied, and the structural change. This eases writing and testing of transformations, in particular if they become complex (like in cases of gapping).

METALSHOP: Tree and feature display



Edit Rules:
Rule (or groups of rules) are loaded into the editor; after editing, they are checked for syntactic wellformedness and recompiled. Also, the rule history is updated.

Delete, Undelete, Revert Rules:
These actions operate on the rule history. Source compare functions show the grammarians the changes between rule versions.

Trace and Step Rules:
There are several options for rule tracing and stepping, operating on the chart. Rules with a certain identification, left hand side category, right hand side pattern, phrase number etc. can be traced or stepped.

Show Rule Tracing:
This is a kind of "post mortem" inspection. Grammarians can follow the rule history of a chart: Why did a rule not fire although expected to do so? Which rule fed which other one? Was a certain rule called at all? Why not?

Draw Node:
Tracing and stepping is supported by the possibility to click on every node mentioned in the trace to inspect the subtree it represents; also, several nodes can be compared, and even their feature decoration can be inspected. This helps in cases of identical trees which differ just in their feature decoration.

Grammar Maintenance:
Again, grammarians can store their changes in private files, and load them at the beginning of a new session. This enables them to work on a problem without interfering the work of other grammarians. When a new release has to be prepared, the private files are merged into a new official version.

Benchmarking:
In order to assure quality, there is a set of benchmark texts which are run every time the grammar changes. All changes in the translations are identified and printed in a delta-file, together with some statistics (number of rules fired, performance, improvements, deteriorations, etc.). This enables the users to see if a change had unwanted side-effects, how big an improvement may be, etc.

Rule statistics:
We also have the possibility to see for a given text set which rules fired how often, how often they were rejected, and how often they contributed to a successful parse. This is important to adjust the rule levelling, and to test if the rules do what they are supposed to do.

6. Further development

The development of METAL is directed towards better integration into the documentation and translation environment. It should become a real Computer Integrated translation (CIT) system, not just a plain translation black box. Much of the success of the system depends on its proper integration into the processes it is supposed to support.

Therefore, some of the next tasks are:

- support more and different editing and desktop publishing systems
- support commonly available hardware platforms
- support more language pairs
- find a lexicon and terminology database which supports both the human translation process and the machine translation; this question is central for the acceptance of a CIT system as in practice, both ways of translation will coexist.

Moreover, we have to refine and further specify the METAL Interface Representation and to make it stable and useful also with respect to new languages which we will add to our spectrum.

7. Literature

G. Adriaens, M. Lemmens, 1990: The Self-Extending Lexicon: Off-Line and On-Line Defaulting of Lexical Information in the METAL Machine Translation System. Proc COLING Helsinki.

J. Alonso, 1988: A Model for Transfer Control in the METAL MT System. Proc COLING Budapest.

J. Alonso 1990: Transfer InterStructure, Designing an 'Interlingua' for Transfer-based MT Systems. Proc. 3rd Intern. Conf. on Theoretical and Methodological Issues in MT Austin, Tx.

W.S. Bennett, 1991: METAL: Past, Present, and Future. Proc of 1991 Workshop on Machine Translation. Taiwan, R.O.C.

H. Caeyers, G. Adriaens, 1990: Efficient Parsing Using Preferences. Proc. 3rd Intern. Conf. on Theoretical and Methodological Issues in MT Austin, Tx.

X. Garcia, X. Rodellar, 1991: ODIF/MDIF Converters. Esprit Translator's Workbench Report

R. Gebruers, 1988: Valency and MT: Recent Developments in the METAL System. Proc. 2nd Conf. on Applied Nat. Natural Language Processing. Austin, Tx

M. Meya, J. Vidal, 1988: An Integrated Model for the treatment of time in MT-systems. Proc. COLING Budapest

Th. Schneider, 1991: The METAL System, Status 1991. Proc. MT Summit III, Washington

J. Slocum, 1981: A practical Comparison of Parsing Strategies for Machine Translation and Other Natural Language Purposes. Ph.D. Dissertation, Univ. Texas.

Gr. Thurmair, 1990: Complex Lexical Transfer in METAL. Proc. 3rd Intern. Conf. on Theoretical and Methodological Issues in MT Austin, Tx.

Gr. Thurmair, 1990: Parsing for Grammar and Style Checking. Proc. COLING Helsinki.