EXCHANGE INTERFACES FOR TRANSLATION TOOLS

Gr. Thurmair,
GMS, Balanstr. 57, D-81541 Munich
gregor.thurmair@gmsmuc.de

Abstract

The following paper presents an overview of current discussions of exchange interfaces in the area of multilingual processing. It first discusses the principles which are relevant for the definition of such interfaces; it then presents a state of the art and a proposal in the area of text interfaces, translation memory interfaces, and terminology exchange. The approach is bottom-up, i.e. it starts from existing interfaces and existing requirements, and intends to be of practical use. It reflects the discussions in current multilingual research projects of the EC, like OTELO and AVENTINUS.

1 Relevance

The topic of exchange interfaces is relevant in practical system development on two levels:

There is a **users'** point of view. Users want to use a variety of tools and components to do their translation job.

- They must take into account that the process of translation involves not just one but several processing components, like a memory, a term bank, an MT system. So the same text must be shared by these components. All of them must understand a common text handling schema, otherwise converters must be written which usually deteriorate both productivity and processing quality.
- They want to share resources between different tools and components: Users want to use the alignment tool of company A but the memory of company B. Users want terms of term bank A to be imported into the MT system B, etc.; exchange of resources is of significant practical importance because resources form the main investment of the users: They set up their specific terminology and their memories, and they want to protect this investment from being independent of the particular vendor of an MT or TM system.

This practical relevance forms the starting point for projects like OTELO ([Quartier 97]), aiming at a common translation infrastructure for different translation components.

There is also a **developers'** point of view: Developers need to integrate different components into new applications. They need to offer different levels of translation capabilities: Machine Translation where the proper language pairs and the proper quality exists; Translation Memories for some standardised text types; Term Substitution in case there is nothing better. On top, they may want to run information extraction or indexing components on a given text. All these components must communicate to each other; so easy text exchange must be fulfilled if a complete multilingual indexing system is to be set up, as well as multi-component access to linguistic resources. This is the starting point for projects like AVENTINUS (Thurmair97b]) which aim at multilingual information processing and retrieval.

1.1 Texts

Texts form the basis of all linguistic components; all of them access a given text. However, most texts are subsets of documents, and documents consist of more than pure text: Non-text sections like graphics and images, references like hypertext links or hot spots, layout information like indents or format types can also be found in documents. Most document or word processing systems have their own way of describing these items, attempts to standardise these issues on a content side were made (Open Document Architecture (ODA/ODIF), ISO 8613, cf. [Appelt90]) but failed in praxi; attempts to standardise on a

syntactic level (SGML/HTML/XML) are currently prevailing but not supported by all word processing systems. Proprietary formats like RTF still play an important role.

Text handling first has to identify relevant document parts, i.e. its text elements. But this is just one part of the task. Text Handling must be considered on three levels:

- We have **input information** for the linguistic processor. This information consists of the text to be processed, but also some formatting elements must be looked at: There is information which needs to be taken into account by the text handling component. Examples are text types, font information, in-text information like footnote markers, anchors for graphics, and others. It needs to be defined which elements must be looked at; here different text handling formats have different proposals.
- In addition to the input information "inherited" from the word processor, we also have processing or **communication information** if several tools run on one text. Assume there is a pre-processing tool which marks up certain text elements, like file names, email addresses, and so on. This information should be passed on to the machine translation or the memory component. A possible way of doing this is to insert some markups into the text. These markups must be understood by the linguistic processor.
- A third type of markup is the **output information,** i.e. the information which the linguistic processor hands over to its environment. Examples are: correction variants in case of a grammar checker; translation alternatives given by an MT system; fuzzy scores found by a memory component; and the like.

A text handling format will have to support all three types of information, not just the input side.

1.2 Resources

There are two main resources in a Multilingual Environment:

- Translation Memories, containing previously translated material
- Terminological Databases, containing terminology with annotations.

Both resources are user specific but not tool-specific. So the resources must feed different components and tools, with different requirements: A term may need detailed linguistic description if used in an MT environment; it may need definitions, contexts etc. when used in conventional terminological contexts. As both types of tools will coreside in future applications, it is either necessary to build resource databases which support multifunctional aspects (cf. [Thurmair 97a]), or to exchange information using exchange formats.

Similar problems need to be faced in the area of translation memories. Not only the definition of segments of the memory need to be considered (even if sentences are considered to be the basic unit there are still differences in sentence segmentation: Treatment of semicolon or colon punctuation is different, and some phenomena are also language-specific (like the period after a number in German indicating an ordinal number). Here again it is necessary to define possible exchange possibilities between memories of different types, to protect the investment into the linguistic resources.

2 Requirements

This section discusses the requirements for exchange formalisms. We must find some criteria which information items we need to take into account in our exchange formats; which linguistic tasks we envisage; and which output we want to deliver. All these factors determine the definition of exchange formats.

2.1 Criteria

Text markup schemata are very numerous, and will continue to be proposed and invented. In order to have a sound basis to define a text Handling Format, we should think of the purpose of such a format. In

our case, the purpose is to support linguistic text processing. This means that we should review the existing formats for information which will influence the linguistic processing of a given text: All markups which have an impact on this must be treated; all markups which do not have influence on it need not be looked at.

To give an example:

- Formats like page and column layout do most probably not influence the linguistic processing of a given text. Markups like banners, icons etc. need not be considered either (except for icons like a return key).
- Text portions, on the other hand, are always relevant from the point of view of linguistic processing. Sometimes it is even interesting to know if a text segment is a heading or not (because capitalisation rules in English are different for headings and plain text; linguistic processing must consider this in order to avoid wrong analysis of the data.

Of course, the vast majority of markups is somewhere in between; so each individual markup must be considered for its relevance for processing.

2.2 Processing

If we talk about linguistic processing, we should be more precise and define which types of processing we want to consider. The following proposal is intended to support the following types of processing (it is presupposed that there is at file-based, i.e. batch version of the respective tool):

Monolingual Applications:

- **Terminology Verification:** A text is reviewed for correct / incorrect use of terminology. Wrong terms, illegal abbreviations etc. are recognised. Input is the text (with its markups), output are some marked terms which need to be displayed by the external editor (cf. [Haller96], [Schütz96])).
- **Controlled Language** (Spelling, Grammar and Style Checking): We need to analyse the text, and insert error messages, correction proposals, and the like. Processing levels are words, sentences, and whole texts ([Nyberg/Mitamura96]).
- **Indexing:** We need to identify good index terms in an input text. Depending on the indexing environment, index terms can be words in full form or in base form, stems, multiwords, etc. We have to mark relevant text portions, or create / insert good index terms into the text.
- **Information Extraction:** We want to identify structured items, like person names, dates etc., as well as complete scenarios (like hiring events, company take-overs) in texts. Intermediate results will be to mark the items found, finally we want to group these items (and their referents) in templates and scenarios (cf. [Cunningham/Gaizauskas/Wilks95]). Many of the structured items need special treatment e.g. in Machine Translation (we do not want to translate *Microsoft Hotline* as *mikroweiche Heißleitung* but keep the proper name information).

Multilingual Applications:

- **Term Substitution:** A text is analysed and compared to a term bank, in order to identify terms and their translations. The translations must be inserted in the text. This is needed if we want to prepare a text for human translation, or if we want to do information browsing.
- **Translation Memory Lookup:** A text is looked up (usually on a sentence level) in the memory. Input information would comprise formatting but also certain text elements considered to be variables (numbers, dates etc.), output would be one or more hits, together with some score how good the match was. In cases where the “bad” text portions need to be translated by the machine, we must communicate to the MT component which units are “good” (i.e. have a good match) and which ones are “bad” (have a bad or no match).

- **Machine Translation.** Machine Translation also takes a text (usually sentence-type units) and produces translations. It needs to interpret input information like fonts and anchors; it needs to interpret communication information like markups of named entities (dates, abbreviations etc.), and it will produce output information, namely translations and maybe variants / alternatives for certain text regions.
- **Translation Quality Control:** Such a component also takes a text, checks if all text portions have been translated, if the official terminology has been used, etc. It uses the same input and output information as monolingual terminology and text verification.

We also have to take into account that in larger projects, there is a variety of such tools which run in parallel on the same input text. An example of such an environment is the AVENTINUS system ([Thurmair97b]).

2.3 Reformatting

It is essential for text handling to know if a text needs to be re-layouted after processing, or not. Tools like Information Extraction do not intend to further process the text; they just want to create a structured information template from the text. Tools like Machine Translation, however, need to deliver the target language document in the same layout as the source language document.

For text processing, applications like Machine Translation require that we have to keep *all* text markups (not just the ones which are linguistically significant) in order to be able to reproduce the target language document in all its layout information. (Note that this sometimes requires to change this layout, e.g. we need to switch the language parameter from source to target language, otherwise the wrong hyphenation and spelling routines will be activated).

There are two approaches to handle re-formatting:

- Either we can extract all the text portions from a document and create a **shadow document,** containing all layout information and just pointers to text portions; the text portions are translated and re-inserted into the shadow file to construct a full target language document. Most text handling formats follow this approach.
- Or we keep all information just in one file and create a **superstructure** containing the original document plus the linguistic extensions. After processing, the irrelevant document parts are stripped off. This approach is followed by the EURAMIS project ([Waldhör97]).

The majority of applications follows the first solution for practical reasons: In case we have to transmit files then transmission of superstructure files is usually significantly more costly than transmission of just text files.

3 Text Handling Interfaces

This section reviews different proposals to text handling, and sketches a merger of the most important features of these proposals.

3.1 Current Approaches

This section gives a very short overview of the current formats; a detailed description and assessment can be found in [Thurmair96].

3.1.1 Text markup approaches

Text Markup approaches have been proposed by a series of authors. All of them follow the shadow file approach, i.e. they identify text portions and leave the other markups aside. The most important formats are:

- the METAL Document interchange format (MDIF) [Polzer91]). It has just one category of text (namely sentences). It distinguishes between fonts, data elements (comprising footnotes markers, anchors etc.), and constants (like acronym identification). Fonts need to be copied and moved with the words they enclose; data elements just must show up in the output text, and constants are communication markups which enclose text plus additional information for the MT engine.
  An extension of MDIF ([Bussé91, Bussé92]) treats paragraphs as basic units, instead of sentences, and allows for hierarchical text organisation; this is needed for alignment. It also allows for description of hypertext links and hot spots.
  The Logos Exchange Format (LEF) is quite similar to MDIF and basically has the same coverage.
- the Eurolang Document Interchange Format (ELDIF) ([Girard93]) goes beyond MDIF: It is fully in SGML notation. Contentwise it distinguishes between main textflow and sub-text flow (text in text, like footnotes: They are often simply inserted in the main textflow, like in RTF or MIF files). The subflow needs to be separated (otherwise the linguistic processor produces errors e.g. in case of pronoun resolution), and must leave an anchor element in the main flow.
  ELDIF was intended to be multi-layered, starting on document level (ELDIF-D), and then creating higher processing levels (ELDIF-L) defining internal linguistic interfaces. For the current purpose, ELDIF-D is the format to look into.
- The Eurotra Document Interchange Format (EDIF) ([Devillers91]). Like ELDIF, it is intended to be multilayered, describing document level, sentence level, and word level phenomena in an SGML based way. This format does not quite consider the input markups; it mainly concentrates on annotations of linguistic units (morphemes, words, phrases): But it distinguishes different types on input text: Standard text, headings, and list items. This is justifiable as headings and list items (many ungrammatical structures!) deserve special treatment by the linguistic processor. Treatment of fonts, text-in-text phenomena etc. was not sufficiently defined for EDIF. A new element, however, is the possibility to represent alternatives, on sentence and word level. This possibility is very useful, even if originally intended to support linguistic descriptions only. Also, some text elements (dates, formulae) could be defined beforehand, leading to some communication information possibilities between tools. The higher level interfaces are specific for Eurotra / ALEP, and have not undergone any standardisation proposals yet.
- The MULTEXT document format. ([Ide95]): This format is designed mainly for corpus annotation purposes. It is a multi-layered, SGML based format similar to EDIF. Like in EDIF, input markups, communication markups, text-in-text phenomena etc. are not very elaborate in the format; fonts are just tagged as such. Texts are divided into four hierarchies (divl to div4) which seems somewhat artificial.
- The SECC document format ([DeBrakeleer-Macken94]). SECC is an LE-project for simplified English and correction of text. The project uses markups for text alternatives, and some global feature-value structures to collect diagnostic information and display error messages for given sentences. The respective sentence portions need to be marked (as one error message like *"wrong compound"* can refer to different constituents in a long sentence).

3.1.2 The TIPSTER approach

As opposed to the markup proposals just sketched, the TIPSTER project ([Grishman95]) proposes a different kind of text organisation. It stores all documents in a database, together with the annotations. Instead of inserting markups into the text flow, and having to parse them, TIPSTER creates a special database for the markups, and describes them in terms of byte offsets (e.g. it has a *person* from byte 4 to byte 7, a *noun phrase* from byte 1 to byte 14, etc.).

This approach has two main advantages over the markup approach:

- It leaves the original data untouched. This allows the original data to remain on CD, or on any device; they can easily be shared with other applications (like retrieval systems) as they remain unchanged.
- It needs not to parse *all* markups when only some of them are of interest. In an SGML environment, the input stream must be parsed every time a certain piece of information is looked for. This is due to the sequential order of the text flow. Such a procedure creates significant overhead in large markup systems and slows down the processing. In TIPSTER, all markups (and the text) can be approached independently of each other in an easy and fast way.

However, in a multilingual environment such an approach has its problems:

- There *must* be a new text which differs from the original one. So there is no advantage in having the source text in a static environment.
- As soon as the text changes, all the byte offsets need to be recalculated. This requires strong updating overhead.

As a result, the markup approach seems to be preferable for full-fledged linguistic processing; the TIPSTER approach has its strength in information extraction applications like TREC and MUC. However, the markups should be restricted and not require a full-blown SGML machinery.

3.1.3 The Euramis approach

Euramis is a project which integrates components for the Translation Service of the European Union. It supports a linguistic resources database, a translation memory, and the connection to an MT system. The tools communicate via a common format, the Euramis pivot format. Details can be found in [Waldhör97].

The most relevant features of the Euramis pivot format are the following:

- The format follows the superstructure approach, i.e. a given document is enriched with additional markups. No shadow document is created.
- The Euramis format is fully Unicode based. Most other formats are tuned towards ISO-8859/1, and use special devices for foreign characters.
- The standard input information markups are covered by using one existing document format, which is HTML (Wilbur) in case of Euramis. All the headings, fonts, and markups defined there are by definition legal Euramis markups. Documents in a different format (like RTF) write their specific information into a specific markup which remains untouched until the original document needs to be reproduced.
- Euramis defines a set of special markups on top of HTML. They are basically used to store processing results: There are markups for TM results (including information on the quality of a match, with and without bonus), for MT results, and for inserted words or phrases from the Linguistic Resources Database. Some of these markups and attributes are rather project specific.
- The communication information is explicitly maintained in the format. Each component (memory, MT system etc.) leaves information which markups it has produced, and in the header it is stated which tools have processed a given document. Even if there is not much interaction now, the format provides the possibility of doing so.

The Euramis pivot format is the most elaborate text handling format at present. It supports all input markups by using HTML as a basis; it adds communication information between the different tools, and it integrates terminology and textual markups in a common DTD. It must be noted, however, that

- the files to be handled become rather large (doubled because of the Unicode approach, and doubled or tripled again by integrating the translation tool results and their variants)
- there is much Euramis specific information which cannot be supported outside of this project.

3.1.4 The OpenTag Initiative

This initiative ([ILE97]) proposed another text handling format. It is SGML / XML based; it follows the shadow file approach, and it is based on markups. It defines paragraphs, some font markups (italic, underline etc.), and some more formats like index definitions, sort orders, conditional texts, etc.
This proposal belongs to the group of text markup approaches as described above, but falls behind many of them as there is no clear principle in the design of the exchange format. Non-text-related elements (like column breaks, conditional texts, or table cells) have been included, others (like headings) are missing. Sort order is considered to be relevant (although it is not a property of a text but rather of a language), text variants, text-in-text flow, hierarchies of paragraphs etc. have not been considered.

The impression of being a rather additive collection of markups is also backed by the proposed extensions, some of which are necessary (like the possibility to give term variants), others are of no importance for linguistic processing at all (like UI groups).

Unlike markup formats like EDIF, however, OpenTag cares about conversion problems from external formats (like RTF, MIF) into the text handling.

3.2 The Open Text Handling Interchange Format (OTHIF)

As a result of the assessment of the different format proposals, the following requirements can be deduced:

1. The structure of the THI should be a kind of ***SGML notation***. We should not use full SGML for performance reasons but just adhere to this standard in case somebody uses an SGML parser for THI files. Otherwise, we should implement a simpler and faster THI parser.
2. The character code should be ***Unicode***. If this is not possible we should specify the character code in the header (and have a device to switch character code). In any case we need a possibility to process alien characters.
3. A document consists of a header and a body. The ***header*** contains parameters related to the whole document. The format of these parameters should be some feature-value notation. We have to specify if there are obligatory and optional parameters.
4. The ***body,*** or text flow, will be such that the text is divided into ***paragraphs;*** paragraphs have attributes (id numbers, types, fonts). Some types of paragraphs are linguistically relevant (like headers); this should be mentioned. We also need an attribute to indicate that a paragraph should not be touched.
   Subparagraphs are not represented as such.
   Out-of-text elements will be represented as an anchor in the main text flow, and as a special paragraph after the main text flow. There will be an attribute of a paragraph indicating whether it is main flow or subflow. This minimises formatting overhead, as well as context problems for linguistic analysis (pronoun resolution etc.) (Moreover, the converter does not know about sentence segmentation).
5. Paragraphs are divided into sentences (or ***text units,*** TU). No alternative sentence segmentation is stored.
   Multi-hierarchy phenomena will not be supported.
6. Text units contain different ***format elements****:*
   - Fonts (need to be distributed to the respective words)
   - Data (anchors, footnotes, pagebreak etc.: need to be kept)
   - Literals (constant, name, date, title etc.). Also index terms and named entities are described here. These literals should be user definable
   - Phrases (mostly because of alternatives, cf. below)
   - Parameters (language change etc.)

7. We need to handle ***alternatives,*** on the level of paragraphs, sentences, and phrases. Alternatives have a type (e.g. *en_translation_of, error_in, alternative_for)* and a reference number (e.g. for controlled language: error number).
8. We should not handle lower level linguistic annotations. We should not do word or even morpheme segmentation (because of ambiguity problems): We need a more sophisticated interface for this purpose. For corpus work, however, we can add an attribute to a phrase marker, giving the part of speech of this phrase.

First the elements of the text handling format are presented; then the attributes of these elements are discussed.

3.2.1 Elements

We can split the markups into two classes: a document level and a text unit level.

On **document** level, we have to order the different alternatives and sub-text-flows of a document. The proposal is as follows:

- The document consists of a header and the text, the header containing some parameters
- The text consists of a sequence of paragraphs, being either main flow or subflow (this is specified by an annotation).
- Each paragraph consists of a sequence of variants, being either language variants (in case of translation) or correction variants (in case of grammar and style checking). This may be surprising as usually, versions of different languages are kept separate in different files, with some alignment being done between them. However, as one of the goals is to produce language variants, this production being supported by different tools, it seems to be intuitive to be able to control this process, e.g. by asking an MT system to just translate sentences where no TM lookup was possible.
- Each textflow consists of text units. Text units again can have variants but do not need a distinction between main and sub flow.

So far we have the following very simple grammar:

Doc	*:: = Header? Body*
Body	*:: = p+*
p	*:: = pv+*
pv	*:: = tu+ | text*
tu	*:: = tuv+*
tuv	*:: = text*

On text level, we should support the following elements:

- String. This is the text itself (#PCDATA).
- PHR: Phrase. This is a part of a text. As opposed to the text, there can be variants for such phrases (e.g. in case of terminological analysis, we would have a phrase, and one or more target equivalents)
- LIT: Literals. Literals are semantically relevant units, i.e. have to be treated as words, and need to be linguistically interpreted. Literals are e.g. names, dates, filenames, abbreviations, etc. Literals need a linguistic description. Literals also cover index terms and named entities for Information Extraction.
- Fonts. Fonts define the formal characteristics of the texts. They are related to the words, i.e. in translation they change together with the words; if a word is translated into several target language words, the fonts have to be copied, etc.
- Data.. Data elements are elements which are in a sentence and stay there (need not be copied or even moved). Examples are footnotes, anchors, pointers to subflow paragraphs (sub_id), and the like.

- Links. Links contain references to a target text portion, attached to a given word. This is like in HTML.

Relevant text portions will be found between all the markups except for the data elements.

Nesting of elements is forbidden. Also, not all markups are allowed to occur in all other markups: Phrases can be within font markups but fonts cannot be within data markups. The following table specifies the respective inclusion and exclusion relations:

Table 1: (DOWN can contain RIGHT)

--	String	Phrase	Literal	Link	Font	Data
String	-	+	+	+	+	+
Phrase	+	-	+	+	+	+
Literal	+	-	-	-	-	-
Link	+	+	+	-	+	-
Font	+	+	+	+	-	-
Data	-	-	-	-	-	-

The reason for this is as follows:

- Strings (str) (i.e. linguistically relevant text portions) can contain all the other elements. They also can consist of just a phrase, or just data.
- Phrases (phr) are marked as far as they have variants. Within those variants, we can have strings but also literals, links, and fonts.
- Literals (lt) are also terminal elements.
- Links (lk) can contain literals and fonts if there are multiword elements to be marked
- Fonts (F) can contain anything except data. Fonts can be nested. In order to avoid combinatory problems (text highlighted *and* italics etc.) we will use the different font types as different markups, just like in HTML, and not as attributes to a general *font* markup.
- Data (D) can just contain #PCDATA, linguistically irrelevant information.

As far as terminology is concerned, we should stick to the HTML conventions for fonts (meaning that we have special markups for bold, italic etc., no common *font* markup); what we call link is called anchor there (A); but anchors have different meaning in document processing, so link is the better term.

The grammar for the textual elements is complicated by the fact that the respective elements e.g. of a link markup can occur in any order, but one element (the string) is obligatory. To avoid ambiguous parsing, we need a complex expression.

Text :: = *(STR|PHR|LT|LK|F|D)+*
STR :: = *chars+*
PHR :: = *phrv+*
phrv :: = *(LK| F)* (STR | LT) (F|LK| STR | LT)**
LT :: = *chars+*
LK :: = *F* (STR |LT)(F|\STR|LT)**
F :: = *TT| I | B | U | STRIKE | BIG | SMALL| SUB | SUP*
TT, I, B, U :: = *(STR | PHR | LT | LK)+*
STRIKE ::= *(STR | PHR | LT | LK)+*
BIG, SMALL :: = *(STR| PHR | L T| LK)+*
SUB, SUP :: = *(STR | PHR | LT | LK)+*
D ::= *#PCDATA*
chars :: = *<some Unicode character>*

As usual in SGML, the elements can have attributes. These attributes will be declared in a separate declaration. They should be defined as close as possible to the units they belong to. Their structure is "<attname> = <attvalue>", the values being atomic.

The different elements have the following attributes:

- On ***paragraph*** level we need a unique identification: every paragraph has a unique identifier. This can be an integer. We also can have font information: a paragraph can have a default font, which holds for all its alternatives. Then we need the paragraph type: as mentioned earlier, some types are linguistically relevant. Such types are: heading, figure, list element. Others are less relevant. Finally, we must have an indication of text flow: A paragraph can belong to the main flow, or be a subflow of some mainflow paragraph. The subflow is pointed at by an indication (sub_id) in the mainflow.
- On ***paragraph-variant*** level, we need just the variant identification (could be an integer), and the language. We may have the same paragraph in different languages. (This allows users to just click on a paragraph if they want to see alternative languages. The data structure could be produced by paragraph alignment). We do not take into account (monolingual) versions of paragraphs, although we could do so.
- On ***text unit*** level, we only have a unit identification
- On ***text variant*** level, we have the variant identification, the language (we can have language variants on sentence level, e.g. as a result of a translation memory lookup), the type of the variant (values being *"correction", "MT-output", "term-analysis-result", "TM-output-perfect", "TM-output-fuzzy"*, and others), and some subtype or flag for additional information, like the *"error-nr"* in a controlled-language environment, an information that MT could not produce a full analysis, etc.

Other information must reach the variant level as well, e.g. the paragraph type information.

This way, we can uniquely identify text unit variants by a chain of paragraph-id, paragraph-variant-id, text-unit-id, variant-id. The alternative is to have just a global numbering for all sentences without being able (or willing) to locate them in the document flow.

On text level, the elements have attributes as well:

- For ***phrases,*** we need no attributes as phrases occur only if there are variants.
- For ***phrase variants,*** we need the following attributes: the language (we can have different languages for phrases), the type of phrase (which can be a term replacement, an error correction proposal, etc.), and a subtype or flag for the respective type (e.g. marking alternatives in l:n translations).
- For ***literals,*** we need the type of literal (e.g.: abbreviation, date, constant, numeral, name, place), and its linguistic category (e.g. noun, adverb) (remember that these elements need to be parsed later!)
- For ***links,*** we need the reference where the link goes to (a paragraph number, or a filename or an URL)
- For ***fonts,*** we need the font type (bold, italic, etc.)

3.2.2 Header Information

Documents can have headers. Headers should specify global information for the whole file. Such information could be:

- TITLE: as title as in HTML. Should be optional, however
- FILE: the filename (? May lead to confusion if the file is copied)
- VERSION: a version number for this file
- CODE: The character code the file is in
- PROCESSED_BY: The tools which have processed the files already. Values are: MT, TM, TA, CL, LNK, similar to the v-types described above.

- CMT: a comment field

The whole header is optional for the time being; it may need to be revised when we start implementing the processing functions.

The complete DTD for the Text Handling proposal is given in [Thurmair96].

4 Translation Memory Interfaces

Exchange Interfaces for Translation Memories are needed if users want to exchange linguistic resources. There may be different reasons for this:

- Resources need to be available in different memory components
- Users want to use aligning components of product A but memory maintenance components of product B
- There is a central memory parts of which need to be downloaded into local PC front-end environments (as in the case of Euramis).

The problem of exchanging memory information can be described on four levels:

- **definition of segments:** The segmentation routines must be identical in the memories involved; otherwise data can be taken over but the new memory software will not be able to activate these segments (or will only have fuzzy matches). Some memories (like ROBIN) are based on phrases, others are based on sentences; and sentence segmentation procedures differ between sentences.
- **order of segments:** Some memories use the fact that a segment has a certain context (i.e. the text flow of a document), others have just a set of segments, without structuring them. Exchange may want to take this fact into account.
- **Annotations:** Segments may be annotated. Annotations are often significant for segment selection, e.g. in case of conflicting translations. Annotations should be object of the exchange.
- **in-text phenomena:** Segments may contain fonts, markups, literals for numbers etc.; all these phenomena may be relevant for the matching quality. They are similar to those to be considered in the text handling area.

In principle, as OTHIF contains text segment variants, it can be seen as a superset of the requirements for translation memories. However, there are special technical side effects which need to be looked into in more detail; they go beyond pure exchange but also concern the functionality and workflow of the TM products involved.

5 Terminology Interfaces

Much more effort than in exchanging memory contents has been spent in order to allow for easy exchange of terminology. Terminology may be sent together with a text to be translated; it may be exchanged between different term banks (e.g. downloaded from a central repository); it may be exchanged between different system components which all have requirements to the lexicon, e.g. from a term bank to an MT system.

The new developments in the integration of many linguistic components also have consequences of the lexicon; pure term banks or pure MT lexicons will become less and less important in favour of multifunctional and multilingual lexicons. The requirements to such a multifunctional lexicon, using the AVENTINUS lexicon database as an example, is described in [Thurmair97a]; the structure of the OTELO database is given in [Ritzke97]. It consists of five sections:

- a central entry section where an entry is defined in its basic characteristics (canonical form, part of speech, subject area tag, etc.)

- a linguistic section where all information relevant for linguistic processing is described; this section is important for all linguistic processors. Information stored here is gender, inflection class, syntactic subcategorisation, semantic type, etc.
- a terminological section where "classical" terminological information is stored; examples are definition, source, contexts, references of different kinds
- a transfer section where translation equivalents, as well as the type of equivalence are stored
- a cross-reference section where links between terms of the same language are given, like *is_synonym_of, has_stem, is_broader_term_of, is_forbidden_term_for*. This can be used for controlled language and for indexing.

As a consequence, purely terminological exchange formats cannot satisfy the requirements of multifunctional lexical databases; some more elaborate exchange functionality must be defined.

5.1 Current Approaches

5.1.1 MARTIF

MARTIF is the proposed ISO Standard for Terminological Exchange (ISO FDIS 12620). It is a further development of the previous MATER and TIF standardisation proposals.

The main characteristics of MARTIF are:

- It is SGML based. All terminological features are defined in SGML notation; there is an extensive DTD to describe terminology.
- It is lexical unit based. Following the terminological view of the world there is a unique notion for everything, having its foreign language equivalents. So every meaning / sense it treated separately. Words with different senses should be forbidden.
- It is organised multilingually. A term can have equivalents in several languages.
- The entries are grouped according to "term information groups" (tig); these tigs contain the real information.
- The real information follows a schema of features and values, describing the formal aspects of the term first, then the content aspects; large effort is spent to administrative issues.

The MARTIF proposal has a purely terminological view on lexical entries: It is categorised according to the term description - concept dichotomy:

- first there is a term description, different ways how a concept could be looked at
- then there is a relationship between the description and the concept
- finally there are relations between concepts.

Such a view may be justified for scientific terminology research; however, it is not sufficient from the point of view of interfacing a multifunctional lexicon. Therefore it cannot be simply taken over for lexicon exchange purposes:

- **Lexical units:** MARTIF also assumes that there is a clear description of concepts possible, in terms of definitions, place of concept in an ontology (which is thought to be a hierarchy) and relationships between concepts, his may be the case in terminology, it definitely is not in common word dictionaries. A simple example is a l:n equivalence relation: This does not seem to be foreseen in MARTIF: Every translation has to be treated as a 1:1 equivalence, coming from a homograph source term. This is obviously not true in common word lexicon environments.
- **Normative Approach:** This drawback can also be seen from the fact that MARTIF is clearly designed for normative purposes. There is a lot of administrative information which will rarely ever

be used by the lexicon exchange data which we have. As a result, some information items are doubled in the exchange formats.

- **Entry Structure:** From a linguistic point of view, the structuring of the entry is strange: Why is the synonym link in group one (term type) where there is a special group on term relations (group 6)? Why are morphological features like *morpheme* (2.3), *gender, number* (2.1) split into different groups? Why is *animate* a special grammar feature while *proper name* is not? Why does *animate* describe the term and not the concept? Similar problems can be detected with the depth of the description: Why is *number* split into *singular and plural* (plus: what about dual in Arabic and Bavarian?) but *gender* is not? Why is *number* mentioned at all but part-of-speech is not? Again, it is not a linguistic structuring but purely a terminological one. This leads to implausible groupings.
- **Coverage:** There are also many items of a multifunctional lexicon (in fact, most of them) which are not present in the description at all. Even if we only take the common features and values as specified in /McCormick 96/, a good deal of them cannot be modelled in MARTIF, let alone the proprietary features of the different systems like METAL or LOGOS or others. This means that MARTIF cannot be used without significant changes.

For these reasons, MARTIF cannot be used as it is; however, it should be possible to support it for the terminological parts of the lexical database.

5.1.2 Other terminological formats

TransTerm

The TransTerm Format, developed by the TransTerm project ([TransTerm96]), adds additional structure to the MARTIF proposal: They extend the proposal in two directions:

- **Hierarchical information.** They criticise that there is no conceptual or hierarchical model in the terminological description of MARTIF but rather a flat information structure. They add the notion of "terminological concept" to the exchange format which is an abstract unit underlying all language-specific notions. Concepts consist of terminological units (terms), and are grouped in terminological containers.
  Such a structure is not really usable in an exchange situation as it presupposes that everybody structures their lexicons the same way. It is also unclear how this structure would link to other types of lexical information which may want to be exchanged.
- **Linguistic Descriptions:** TransTerm adds linguistic descriptions using a construct called Linguistic Representative as an intermediate object between terminological and linguistic unit. For linguistic descriptions, the user is referred to the GENELEX model.
  However, the link has not been described in an integrated and usable manner in the TransTerm reports; the two sides seem to be rather unconnected to each other. Moreover, the GENELEX model does not comprise all information necessary, e.g. there is no multilingual linguistic description in the documents referred to.

So the merit of TransTerm is to argue in favour of including lexicographic information; the proposal itself, however, is not in a state to be simply taken over.

Interval

Interval Interchange Format (IIF). The Interval Interchange Format, developed by the LE-Project INTERVAL, aims at producing a tool to verify terminology; in order to do so, it is assumed that terminology will be submitted in a certain format, the Interval Interchange Format (IIF).

Files in IIF consist of a header and a body; the header contains information on the character code used, and the language tags used). The body consists of entries.

An entry consists of a concept part and a term part. The concept part specifies the project to which a term belongs, a subject area, a concept-reference, or a comment. It is optional. The term part consist of

groups of feature-values. Each group has either a TERM or its EQUIvalent, plus additional features: the string; a type (term, abbreviation, phrase); a reference; part-of-speech information; definition and/or definition-reference; note and/or note-reference; context and/or context-reference

These features are fixed in number, and also fixed in order. A term can have several language equivalents (same language, or different languages). The notation is SGML. INTERVAL does not use MARTIF for reasons of complication although the IIF seems to be a proper subset of MARTIF. They claim that the IIF is MARTIF compatible.

What should be kept from this proposal is the idea of a header information (defining character codes and languages). Otherwise it is clearly restricted to (a subset of) terminological use, and too restricted for our purposes. Also, it standardises the content of the interchange format (in terms of features and values) which is difficult in the area of the lexicon.

5.1.3 Machine Translation Exchange Formats

METAL Lexicon Interchange Format (MLIF)

The METAL Lexicon Interchange Format MLIF ([Polzer91b]) allows to exchange monolingual and bilingual files. It must be known off-line which languages are involved. The character code is ISO8859/1, with escape possibilities for alien characters.

An MLIF file contains of entries, separated by a separator; each entry consists of a set of feature-value information (need not be ordered). Features can have types *(source, transfer, target),* partly as keywords, partly as prefixes. Features can be legal METAL features as declared in the lexicon declaration files.

- Values can have different types: Strings, (characterised by quotes),
- sets or lists, enclosed in brackets, the elements can again be strings or symbols or lists; the order is sometimes significant, sometimes not, depending on the feature declaration in the specifications files
- symbols (everything else; symbols are checked against the lexicon specifications files)

Any legal feature can occur, typed for source or target language, in any order. The MLIF files are not multilingual but bilingual, and they contain only 1:1 transfers. If a l:n transfer has to be specified several entries forming one package have to be written. They also are directed (de-> en differs from en->de).

A positive element is the basic structure of having just a flat file of features and values. Also, typing of features needs to be done (assuming e.g. the *number* feature both in source and target must be distinguished). Problematic is that the files are not self-explanatory (languages involved must be known outside, usually by filename) that the character code is somewhat restricted, and that no nesting of features is possible.

Logos Exchange Files (LEF)

The LOGOS exchange files consist of a header, end a series of entries. Character code is ISO8859/1, files are bilingual.

The header contains: Definitions of separators and other markups (delete, modify,...); defaults and global definitions; entry definitions (declaration of legal features and values, and the data type of the respective features: string, integer, different types of data); mappings of external representations into the LOGOS representations.

The data section contains entries. They are organised like a table, containing only values; the feature can be computed from the respective column. So each entry is a record where values have fixed positions.

The format supports field inheritance (i.e. values from the previous entry are inherited to the new one); it also supports nesting of features. Features and values must be compliant with the LOGOS specifications.

The format can also mark the origin of a feature (Logos, XL8 etc.) as well as the language (source, target, transfer).

What can be learned from this is the feature declaration part in a header, and the possibility of supporting feature groups. As LEF is not designed as a pure interchange format many processing-specific items can be found (like origin of a feature, mapping instructions, and the like); also the format is not completely declarative (there are features like *Black Hole Pointer,* or *Logos Overflow 3B* which should not be subject to exchange).

From the point of view of architecture, we should separate the exchange format from the import functions: Mapping, deleting etc. are issues in import as they depend on the software components involved. The origin of a file should be noted in the header of the exchange format. Feature inheritance should be supported only on a global level, otherwise the sequence of the entries of an exchange file becomes meaningful.

5.2 The Open Lexicon Interchange Format (OLIF)

OLIF (Open Lexical Interchange Format) is a proposal which tries to support exchange on a pragmatic level. It is designed to support multifunctional databases.

1. Given the fact that the different exchange proposals differ more in their structuring of the entries than in their terminal elements, OLIF follows an approach to describe just the terminal levels of the entry information, and be as atomic as possible, just giving feature-value pairs.
2. We must support different levels of conversion:
   → Renaming of features or values is the simplest case: *Part_of_Speech = Noun* can easily be converted into *Category = Substantive.*
   → Regrouping of values is another case: *Proper Noun* may be a value of *Syntactic_Type* in one system, a value of *Semantic_Type* in another. There are more complex cases such that certain clusters of feature-value pairs correspond to another cluster: Nominalised Verbs in German may be described as *Category = Noun* plus *derivation = deverbal,* but there may also be a possibility to have a *Category = NomVrb.*
   → Finally, there may be completely different approaches towards certain phenomena; good examples are inflection behaviour or verb argument structures. When comparing the METAL and the LOGOS implementation, it turned out that there was no possibility to have a simple feature conversion.
   For the exchange, the only remaining possibility is to go back to the data level, and just describe this data level in the exchange interface. The different systems must derive their way of description from these data.
3. Defaulting: As linguistic information is particularly difficult to code, and most important for every computational treatment of a text, a strategy has been followed in OTELO to agree on the minimal set of annotations which the single systems need. This minimal set can then be enriched by powerful defaulting components which produce full-blown linguistic descriptions for an entry. This strategy is the more successful the larger the lexicon is: If all the exceptions are stored in a lexicon, defaulting strategies are rather straightforward.
   So the interchange needs just the basic information items which cannot be defaulted by the respective systems.
4. We need to support exchange of both monolingual and multilingual information. This is performed by defining an entry by its monolingual properties, and describing its translations as a link to another (monolingual) entry.

OLIF does not claim to be the universal platform for exchanging all kinds of lexicons. Instead, it tries to allow users to define what they want to exchange, and offer a simple platform where different term banks and lexicons can share their terminological resources. This means that the systems will have to write converters between their proprietary formats (like MLIF or LEF) and the OLIF platform. It will be evaluated in the OTELO project if such an approach is feasible in praxi.

5.2.1 Entry Definition

Following the structure of the database, we propose that we interchange one (monolingual) entry per time, i.e. we interchange German entries, English entries etc. (An entry is defined by a unique Entry_ID, so everything with this entry_id is taken). This differs somewhat from the "classical" terminological entries which interchange multilingual entries (i.e. concepts with denotations in different languages): They would connect entries with different Entry_Ids (as the English part of an entry would have another entry_id than its German equivalent).

In our case the exchange object is basically a monolingual entry, with links to other entries. So, again, all information belonging to one entry number is grouped for exchange. Entries with different entry numbers, e.g. a German entry and its English counterpart, will be two different exchange objects.

Each entry must have a certain number of features which are obligatory. This is necessary to identify the entries. The obligatory features must be available in (possibly) all the lexicon systems which are to be interrelated by OLIF, and must be context-independent (i.e. should not depend on some internal keys and numberings as those may change when entries are exchanged). We could use the following features:

canonical form - language - part of speech - subject area

They will be used not just to identify an entry but also later on to address the target entries for the link features.

5.2.2 Features and Values

The basic information items in OLIF are features and values, separated by a separator sign. The OLIF format should be as flat as possible, and essentially be a list (or catalogue) of feature value pairs. This list should contain the basic (or atomic) elements of a lexical entry.

The reason for this is that we do not know how the target systems with which we want to interact organise and group their linguistic material. Like in Eurotra-10, we consider it to be a sound strategy to give just the terminal elements which then can be interpreted by the different conversion programs.

Therefore, unlike in many linguistic theories, features will have only *atomic* values. No complex or even recursive features (i.e. features which have again feature-value-pairs as values) are allowed.

Features are of a certain *type.* In order to differentiate between different feature types, we should have the following conventions:

- Features can have a type *string* or a type *member.* Member features have values taken from a pre-defined set of possible values, whereas string features can have any string as a value (like *Canonical_Form "Haustier").* Examples for string type features are: *Canonical_Form, Definition, Example,* etc.; examples for member type features are *Part_of_Speech, Entry_Type* etc.
  In the definition of the entry structure it is stated which features have which types.
- features can be single valued (like *Entry_Type)* or multivalued (like *Semantic_Type).* Multivalued features have parentheses around their values whereas single valued features do not.

Integers and dates are treated as members and are converted internally, by the participating filter programs.

5.2.3 Feature Groups

Although OLIF should be as flat and atomic as possible, it is necessary to form feature groups from single feature value pairs. The reason is that some features themselves have attributes:

- A transfer feature can have attributes. It does not just say that the English target term for house is *(house, Noun, GV)* but also that this link has the properties of being *equivalence_type full, transfer_tests none,* etc. This adds some complexity to the interchange format as we have nested structures and not just flat feature value structures.

- A cross-reference link feature can have attributes, like the frequency of that link
- Also, stem variants, or allomorphs, need to be grouped as they indicate differences in number or inflection class

So we need a means to collect features which belong together into groups. We will have four feature groups:

<ALLO> collects all the features which belong to a certain stem variant (table LIN_ALO, LIN_GPMU). <MONO> collects all monolingual features which belong to an entry (Central_Entry, LIN_MoSynSem, Term). <XFR> collects the transfer information. <XREF> collects the cross-reference entries.

Like <ALLO>, we can have several instances of <XFR> and <XREF> groups. Note that they need not be unique (an entry can have e.g. several translations into *English,* or several *Synonym* links).

5.2.4 Specification of target entries for links

We need to specify the target entry to which a link (be it monolingual or multilingual) points. This is not a big problem in any term bank internally as we have unique ID numbers for these targets. However, this relationship does not exist any more in the interchange format (as the ID numbers may change in the different lexical systems involved). So we need to "describe" the targets of the links as exactly as possible: Assume we have an entry

Schlüssel ->RELATED_TO -> Schloss

and we have two entries called "*Schloss" (lock* vs. *castle)* then we need to specify which entry called *Schloss* is meant in the link. We cannot use the entry_id for *Schloss* (which internally is used) as the entry_id will differ in the different lexicon systems into which we want to go. We also cannot use some meaning_variant_number as the target lexicon systems have just one, but also three or four, entries called *"Schloss";* so the meaning_variant_numbers are different for the different contexts in the different lexical databases. So we need a context-independent specification of which entry we mean. The same holds for transfers, of course, if there is more than one target language entry.

OLIF proposes to describe the target entry of (monolingual and multilingual) links by the obligatory features:

canonical_form, part_of_speech, subject_area, language

In principle, we need to describe the *meaning* of the target entry. Devices to describe the meaning, however, are not easily available: Meaning variant numbers in the database are context-dependent and therefore cannot be used. Features like *Semantic_Description* (which also would identify the meaning of a term) may not have been filled out by coders, i.e. cannot be used uniquely. So we can only use obligatory information (which always must be there), and entry-specific context-independent information. The proposal to use subject areas also has the advantage that most external systems (Logos, T1 etc.) use this information as well, and it is a "quasi-semantic" type of information.

However, this procedure may still lead to ambiguous target references; it is the job of the import tool to resolve these references (there may also be no reference whatsoever, e.g. if a synonym specified in the SYN link does not exist in the external target lexicon).

5.2.5 Header Information

So far, we have defined OLIF to be a set of entries; entries are divided into feature groups, feature groups consist of feature value pairs.

However, we also need to specify information which belongs to the file as a whole. This information should be put into the header. So the whole OLIF file consists of a header and a body, the body containing the entries.

There are two types of information in the header: Administrative information, and lexicon specification information.

- **Administrative Information:** We want to deliver administrative information in the header, like: who created the OLIF file and when; which Lexical Database Server wrote it; the character code used, (and possibly more information).
- **Specification Information:** We need to define the features and values used in the body of the file: Users can modify features and values, e.g. add some terminological features, add new languages, add cross reference link types, etc. So we cannot assume that there is a fixed set of 24 features and 87 values which we need to support.
  This means that we have to verify if we can import/export certain types of information. To give the conversion programs a chance we will use the header to specify which feature groups with which features and values will be used in the following body of the file.

So the specification information gives the feature name and all values which this feature can take in the file:

5.2.6 BNF description

So we have the following description of the Interchange format: An interchange file consists of a header, containing administrative and definition information, and a body containing the entries. Entries consist of feature groups. The MONO feature group is obligatory; it is a monolingual identification, containing at least the obligatory features (canonical form, language, part of speech, and subject area). Allomorph, cross reference and transfer feature groups (there can be none / one / many of them) consist of a header keyword and a set of feature value pairs where again canonical form, POS and subject area (for xref and xfr) of the target are obligatory, and optional feature value pairs may follow. Values can be single or multivalued, and can be string or member type.

OLIF :: = header body
header :: = admin_info specification
admin_info :: = authorinfo fileinfo charcodeinfo serverinfo
specification :: = feavaldefs+
feavaldefs :: = “<FeaDef>” (deffeature defvalues)+ “</FeaDef>”
deffeature :: = *(a feature name)*
defvalues :: = defval | “(“ defval+ “)”
defval :: = memberval | “string”
memberval :: = *(a member)*

Body :: = entry+
entry :: = Monogroup Allogroup* Xrefgroup* Xfrgroup*
Monogroup :: = “<MONO>” definition (feature value)*
definition :: = entry lang POS SA
entry :: = “CAN” *(the canonical form of an entry)*
lang :: = (*a defined language value)*
POS :: = “POS”(*a legal part of speech value)*
SA :: = “SA” (*a legal subject area value)*
feature :: = (*a legal feature name)*
value :: = val+
val :: = memberval | stringval
memberval :: = (*a legal value name)*
stringval :: = (*a string)*
Allogroup :: = “<ALLO>” (feature value)+ “</ALLO>”
Xrefgroup :: = “<XREF>” tdefmition “</XREF>”

Linkjype :: = *(a legal xref link name)*
tdefinition :: = entry POS SA
Xfrgroup :: = "<XFR>" tdefinition "</XFR>"
Language :: = lang

Details can be found in [Thurmair97c].

6 Conclusion

In future applications, it will be necessary to merge the exchange formats just described into a common description as the boundary between terms and (memory) phrases and between memories and (multilingual) texts become less and less clear. Such a common description has been proposed by EURAMIS where all resources are described in a uniform format.

7 References

1. Appelt, W.: Dokumentaustausch in offenen Systemen. Heidelberg 1990 (Springer)
2. Bussé, P.: MDIF-II: Design Issues. Liège 1991
3. Bussé, P.: MDIF-II Format description. Liège 1992
4. Cunningham, H., Gaizauskas, R.J., Wilks, Y.: A General Architecture for Text Engineering (GATE) - a new approach to Language Engineering R&D. Research Memo. Univ. Sheffield. 1995
5. De Braekeleer, Gert, Macken, Lieve: SGML Design Study. SECC Report. 1994
6. Devillers, C.: EUROTRA 6/3, Text Handling Design Study, Bruxelles 1991
7. Girard, O.: Spécification du Format ELDIF-D. Site, Paris 1993
8. Grishman, R.: TIPSTER Phase II Architecture Design Document (Tinman Architecture), V.152. 1995
9. Haller, J.: 1996. MULTILINT - A Technical Documentation System with Multilingual Intelligence. In: Proceedings of ASLIB Translating and the Computer 18, London, England.
10. HTML 1996: Web Design Group: Introduction to Wilbur. 1996 Http://www.htmlhelp.com/reference/wilbur/alltags.html
11. Ide, N., Durand, D., Priest-Dorman, G.E., Véronis, J.: Corpus Encoding Standard. (Version B) Multext Report, 1995
12. ILE97: Open Tag Initiative. Www.ile.com. 1997
13. ISO FDIS 12 620: Terminology - Computer Applications - Data Categories
14. Nyberg, E., Mitamura, T.: Controlled Language and Knowledge-based Machine Translation: Principles and Practice. Proc. CLAW 1996, Leuven
15. Polzer, R.: MDIF - Description of the Format-Independent Interface to Machine Translation. Siemens Nixdorf, Munich 1991
16. Polzer, R.: MLIF - METAL Lexicon Interchange Format. METAL Technical Report, 1991 (91b).
17. Quartier, P.: The OTELO Project. Proc. MT Summit 6, San Diego. 1997
18. Ritzke, J.: Common Lexical Resource Format: Format Specifications. OTELO Report. 1997
19. Schütz, J.: 1996. Combining Language Technology and Web: Technology to Streamline an Automotive Hotline Support Service. In: Proceedings of AMTA-96, Montreal, Canada.
20. Thurmair, Gr.: Text Handling. OTELO Report. 1996
21. Thurmair, Gr., 97a: Ein multifunktionales Lexikon. Proc. GLDV Leipzig 1997
22. Thurmair, Gr., 97b: Multilingual Information Processing: The AVENTINUS System. Proc. FBINAA Conf., Berlin, 1997
23. Thurmair, Gr.: OLIF - Open Lexicon Interchange Format. OTELO Report 1997. (97c)

24. TransTerm Consortium: Final TransTerm Report. 1996
25. TransTerm Consortium: A General Presentation of the Conceptual Data Model. TransTerm Report. 1996
26. Waldhör, Kl.: Euramis Pivot Format. 1997