How to ease translation of programs and documentation

Klaus Daube, IBSI CASE Development AG, Schwerzenbach/Zürich (S748)

Multilingual applications need more design and planning effort than uni-lingualones. Session 3.4E at SHARE Europe SM 94 describes some of the problems and solutions, not only how to write documentation so that it is easier to translate into several languages. This talk summarises personal experience as well as general knowledge.

The paper is organised in the following sections:

- Program versus documentation?
- Text in program - various aspects
- The basic principle: divide et impera
- English - the lingua franca of informatics
- Terminology saves time and money
- Graphics are not that universal at all
- How to ease the translators job
- Checklist and bibliography

The Author. As a mechanical engineer I started my activities in informatics with Fortran programs to ease my engineering. Support of technical programmers called for documentation of the available services. This led to side tracks on character and code standards, hyphenation, typography and design, which became more and more the grounds of my activities.

SHARE Europe was my first educator in the aspects of host-oriented document processing. The hostile environment of IBM systems called for a Taskforce about Character and Code problems. The resulting White Paper of 1985 was not properly addressed by SAA, so I initiated a new Special Technical Working Group about a National Language Architecture. The White Paper of 1990 was not only spread within IBM, but was recognised also by other vendors and international standards bodies.

Program versus documentation?

In the good old days of programming documentation was ‘considered harmful’, as the programs were written from gurus for gurus. This has changed dramatically. Documentation now is as essential as the application itself.

In an ordinary software project the amount of money spent in development, maintenance and (traditional) documentation is about equal. Even some managers of IS-projects know this. Nevertheless, the effort for documentation is very often forgotten in project plans. Nearly always forgotten is the amount of work implied for translations. The growing impact of international markets also for small companies call for serious considerations in this area.

Modern programs do not need documentation at all.

Some people argue that modern interactive programs on workstations with menus, dialogue panels, sophisticated messages and application assistants do not need any documentation. Looking at the complaints about software piracy seems them to be right. However: the growing number of secondary books on software prove them wrong.

The other side of the medal is this: Help systems, tutorials and application assistant, sophisticated messages etc. are just another form of documentation - not a substitute for documentation. These new forms need much more effort than ordinary paper documentation - if the capabilities of these forms are really exploited.

In addition, paper documentation will be needed at least as an introduction - and as a marketing instrument. As a user of software (not talking about nice goodies from the SHAREWARE shelf) I want to be informed before I dive into laborious installation procedures and other cost.

New forms of documentation need much more project coordination than that of batch programs.

If in the old days one could consider to start documentation at the end of the development cycle, this is not possible at all with interactive programs. Context sensitive help needs good planning and cooperation between writer and programmer. Both must start from specifications, which must be as complete as possible.

In human life nothing is perfect. The specifications need modifications not only after test, but very often also during the development phase. Project control is much more critical than it was for linear batch programs. Interaction between designers, programmers and writers is much deeper. It is also a source of many quarrels, as programmers sometimes do not understand the stupid questions of the writer. But he is the advocate of the user!

The whole process of developing software and documentation became recursive. The shortened development cycles press all actors in this game. So we

- need to do it right from the beginning on
- have rules, guidelines and tools available

Poor or even missing specifications to start from is one of the reasons, why our mother company got into deep trouble and I will loose my job.

Text and program - various aspects

Any process or program comprises three activities: input, processing and output. These may be performed sequential as in batch programs or in random order as in interactive applications.

Input data uses conventions, which may not be known to a program.

Input data can not be processed directly in many cases, as they are not in the same form as used in the internal storage. Hence a transformation is needed. Most data does not contain any indication of their nature, for the computer these are just sequences of bits. There are assumptions by the program (better: by the programmer), what data comes in and will be processed. If these assumptions do not hold, the process will fail.

Textual data have the advantage, that a human reader of the source can make an educated guess about the format of the data (for example, its code), because textual data contains redundancy for the human reader. Finding `K BNHAVN` in his Eurocard bill, he is able to fill in the missing `Ø`. In most cases a program will interpret this as a blank...

Comments in input data allow correct interpretation of data which have little redundancy.

Most input data, which is typed by some poor lady (there are only few male data typists around the world) is written once, and then read many times. If this is ordinary text, then there is no problem in understanding the nature of the data. But if this is tabular data, comments ease interpretation. Hence any good program design allows for comments in all input.

We will see later, that all text belonging to a program must reside on files. Hence the comment becomes a vital instrument for translations. Assume, for example, the following parameter file:

TABLE 1 Sample data file with/out comments

Without any comment	With comment
`IBM`	`;general properties of input filter`
`;%`	`IBM  VendorName`
`9999999999`	`;%  Tag-introducer`
`O SOURCE  to be handled specially`	`9999999999  End-'tag'`
`O CUSTNAME  to be handled specially`	`;definition of tags to read listed REQS`
`O UGREQNO  to be handled specially`	`;0 for text contents, -1 for value`
`O UGSTATUS  ReqStatus`	`;¦ tag-name`
`O UGPROUSE  OrgComments`	`;¦ ¦  name of data base field`
`O UGSUBDTE  ReqSubmitDate`	`O SOURCE  to be handled specially`
`O UGVOTES  OrgVotingData`	`O CUSTNAME  to be handled specially`
`O PRIORITY  ReqPriority`	`O UGREQNO  to be handled specially`

Please note the subtle difference of meaning of the semicolon. It introduces comment only on the first position of the line. This allows to have a data item starting with a semicolon, if it is indented.

Comments in message files are essentials for proper maintenance and translation. It supports the answer to old question of programming: Where do I come from? Very often we need to tack the source of messages.

Comments are also useful in output data. Do not assume, that only printed or displayed lists are read by humans. The output of one program becomes the input of another!

Tagged data provide advantages not only to the human reader.

Adding some sort of headers to the data to identify sections, this is called tagging data. We are still lacking general rules to specify input data. Tagged data provide a number of advantages - also for the possible process of translation:

- Natural means to structure lengthy files.
- A search process can speed up to the desired section.
- Information which part is translatable can easily be applied.
- A human reader as well a program can be guided how to interpret the data.

Some established forms are:

- INI files in MS-Windows.
- Document Composition Facility (DCF) tagged input.
- Marked up text according to SGML.
- Dialogue Tag Language of ISPF and OS/2 PM.

An example from the initialisation file of the before mentioned Requirements Management program illustrates the possibilities of tagged data:

```
[ProjectNames]
NumberOfProjects=10
*.......Project title..........Project codes (1st = CURRENT) including Div code
proj1=Programming Languages, ALANG, AAPLX, LAPLX, LAPLO, APLXI, AFORT, LFORT
proj2=End User Support     , MUSER, MDOCP, MOAUT, AOAUT, MICIC, LASAS
proj3=Personal Systems     , MIPCP, AESES
proj4=Tools and Interfaces , ATOOL, AGRAP
proj5=Data Base            , MDABA
proj6=Systems Management   , MSYSM
proj7=AIX and Open Systems , SAIXX
proj8=MVS / JES2 / JES3    , SMVS1, SJES2, SJES3
proj9=Networking           , SNETW
proj10=VM of all flavors   , SVMVM
* For the generation of lists from the data base also OLD Project codes must be used
* in the query. Hence changing the project structure may only add to the list of
* codes, NOT REMOVE any!
* Project titles may become 'Technical areas' in the future
```

Graphical user interfaces add further complications to the use of text in programs.

GUI's contain many textual elements. Besides the coding issue to be discussed later, the length of text in various languages is a crucial issue. A good layout of a dialogue box may be destroyed completely, if the text from the translation is just filled in. The following occurs very frequently:

- Text of labels (names of entry fields, captions on buttons) and menu items is truncated. For example the short English words `Print` and `View` become `Impression` and `Visualisation` in French.
- Elements of the GUI overlap and are rendered unreadable.
- Text entry of (for example file names) is not possible to the desired length.for example, postal ZIP codes may be longer than just 4 figures...

If you do not provide a solution to these problems, the translator will need to invent abbreviations or choose less effective translations.

Coding is one of the most crucial issues for data.

Be aware of the fact, that even in the PC and workstations arena US-ASCI is not the only coding standard in use. Our industry's current habit of encoding characters in 8 bits restricts any coding scheme to at most 256 symbols. For the alphabetic part of these codings at most 191 places are available - not enough to cover only so called 'Western Languages' - not to speak of supporting Eastern European or African languages.

It is very delicate to handle text, if input and output do not use the same coding. with the advent of Client / Server applications even mainframe people are now confronted to this threat. You can not assume, that a certain coding of input data is guaranteed. If your company acquires new market areas, you will suddenly be confronted with these problems.

ISO 10646 is the only cure to the coding problem of text.

Current 'solutions' to the problem like the OEM code page used in MS-Windows are just crunches, as just another coding is used, not more codes are available. The only cure is a universal character set, using at least 16 bits or even 32. Applications are now emerging using the UNICODE standard - the only known implementation of the ISO 10646 standard.

ISO 10646 allows to code characters of all living languages including Chinese and a huge number of mathematical and other symbols. There is even space for private use, for example, for hieroglyphics. A growing number of applications, even program environments, are using UNICODE. For example MS-Windows NT is reported to use it. A Program Development Environment called Galaxy also uses it.

Help systems and tutorials may be treated as closed applications.

At first view, help systems can be seen as closed applications. For example, on MS-Windows help files are interpreted by the Windows Help engine. This view is, however not true on other environments. The creation of hypertext applications like help systems need much more effort than a sequential text of same amount. Hence there is need to use the same input on various environments. Just think of transporting a Workstation help to an EBCDIC environment (with lack of umlauts and accented characters...).

Scripting languages are a source of trouble. Which elements are subject to translate, which not?

Even interactive programs need some sort of scripting language to automate repetitive processes. These scripts are always stored outside of the processor, hence may use different coding than what the interpreter assumes...

Thinking about translation of the application, the very crucial decision comes: which elements to translate and which not. In my opinion it is not good practice to translate well known programming terms like `IF` or `WHILE`, as it is done in the

MS-Word international versions. It will also produce much headache to translate calls to internal subroutines and functions.

When designing a scripting or macro language, take either of these routes:

- Keep everything in English, as these capabilities are most time not used by the so called end-user. Macros and scripts need skills, which are normally only available to programmers - which have a long tradition of reading English.
- Before processing the script, transform the external representation into an internal representation (like, for example, MS-Excel does). Then the interpretation of another language is a matter of a new table.

The basic principle: divide et impera

From the discussion presented so far, it should be easy to draw the following consequences for programming: *Externalise all textual elements from the code*. In detail that means to

- Ban text constants from code. At least collect them in one and only one global module, which can be handled by a translator.
- Do not combine pieces of text to messages. Always use complete message texts, which can be understood and translated.
- Provide a mechanism to edit messages with variable information. The sequence of inserted elements (for example, file names, number of something) is not the same in all languages.
- Allow comment in input data, in particular message files and initialisation files. This helps the user to modify them and the translator to understand.
- Provide comments also in output files. Be aware, that in many cases generated output will be fed into the same program as input again. This method allows to update bulk data with arbitrary tools without the need to interactively fiddle around with individual entry fields.

Message editing, not combining, is an essential principle.

Combining messages from pieces of text is a bad programming habit. Some people still believe in saving storage this way. However,

- for the translator it will be very difficult to find adequate translations to incomplete sentences
- Composed messages most time do not conform with the grammar of the foreign language.

The advice is to use always complete messages with place holders for variable information:

```
Do you really want to replace file %1 with %2?
Wollen Sie die Datei %1 durch %2 ersetzen?
```

Do not make any assumptions concerning input data.

Programmers (at least those grown up with main frames) like to fiddle around with bits and bytes. They often forget the purpose of the application. Hence they are tempted to assume that, for example:.

- only upper case characters appear in the input data.
- no diacritical characters (é, Ä etc.) are used in the input.
- the application will be used only in the country of the developer.
- the keyboard will not provide characters like § or ¦, hence they can be used for program internal purpose (the lazy parser).
- all input uses the same code page (coding scheme).

These issues very often are not addressed in an analysis or design. Hence the programmer feels free to implement his ‘very quick and elegant algorithm’.

Use menus with choices rather than complicated messages with just YES / NO answers.

Most messages are designed by the programmer, who is familiar with the behaviour of the program, but has not the skill of a trained writer. Messages like the following are difficult to understand and to translate:

```
Confirm mouse operation: Are you sure you want to copy the
selected files or directories to C:\FRAME.40\DICT? Press
ENTER to accept, ESC to cancel.
```

Is the question mark a DOS wildcard character? Have I selected just files? Or directories as intend? I need to read the whole message to find, that I have chosen (due to poor user interface) copy rather than move... Why not

```
Confirm copy of selected files from G:\FRAME.40\DICT to
C:\FRAME.40\DICT                          [YES] [NO]
```

Use services of the operating system or API's where ever possible.

It may be challenging, to create the shortest quicksearch program to access items in an ordered list. However, when the language of the data changes, their ordering will not be according to the coding values any more. Only the 7-bit ASCII sorts according to the binary value (and not correct at all). Even in EBCDIC the binary order is absolutely useless.

Having, for example, different sorting rules in different applications on one platform is more than annoying to any user. This can only be avoided by using the functions of the run-time environment. Look out for true national language support. Up to now only POSIX conforming UNIX and System 7 on the Mac provide this. Probably also the Language Environment 370 from IBM also.

English - the lingua franca of informatics

Much of the development of informatics is performed in the US. Large vendors have their roots in this country. Hence much terminology in informatics is in (US) English. Only France and Germany made and make attempts to define an own terminology. Generally speaking, English is the lingua franca of informatics.

English text in informatics is not only read by native English speakers.

Around the world many people have English as their second language. They read English manuals to be up to date, avoiding the translation delay. They become familiar with English terminology and do not understand the terminology of their native language in this particular field. But - they are still not very skilled English readers and have problems with English idiomatic language.

Although the upcoming EU Guideline Machines requests documentation in the language of the country of use - it is not yet certain, whether this will also apply to computer programs not related with the control of a machine. For example, does a CASE tool need Greek documentation, when delivered to Greece? At least we must be prepared for this!

Controlled language is needed for non native English readers.

Non native English readers mostly do not struggle with newest terminology in their field of application, but with the subtlties of stylistic good English. A common trap are ‘False Friends’. These are words, which are spelled similar, but have different meaning. Such English words are, for example: eventual, decent, oversee, probe, tint and welcome.

Controlled language is a means to get better understandable text, and to cut the time for translations. Elements of this control are words, sentences and the whole text, as well as punctuation and layout. The CATERPILLAR tractor company were the first to use controlled language (probably due to less educated users of their products).

Airbus Industries uses controlled language to harmonise documentation on the language skill level. It is used in the six partner companies and about 1400 subcontractors. However, it is not that easy to define a controlled language. It is also not obvious to use simple language for complicated devices...

Simplified English avoids translation at all.

Simplified English uses a drastically reduced vocabulary. It is defined for the International Aerospace Maintenance Language and uses only 786 words. In addition to these a defined technical vocabulary is used.

The goal of this method is to avoid translation at all. These documents are needed in a large number of different countries, but in very low quantity. So the effort for translation could not be justified.

Passive forms and sequences of substantives create ambiguity.

To be neutral in expression, passive forms are used very much in academic papers. Technical documentation also try to be neutral (see this paper). However, both for the reader (user of the product) and the translator it is not always clear, whether this expresses a fact or requests for action.

When writing in German, heavy use of substantives create long combined words. With these it is often not clear, what they really mean. Equivalents in English are sequences of substantives like ‘keyword parameter’. Is this a parameter *in the*

form of a keyword or a parameter *to the* keyword? Here one of the golden rules of writing (to be short) must be left and additional words must make the meaning clear.

While this might be considered just an instance of clear writing, it is a problem in translation. Both in English and German the substantives do not define the grammatical time. A merge file (misch-datei) is the file being merged *and* the merged file. But in French *fichier à fusionner* is not the same as *fichier fusionné*.

Terminology saves time and money

Terminology assures that concepts and ideas are communicated correctly to the reader of manuals and among developers. Sometimes terminology becomes a shorthand notation in a field of knowledge. Hence it is vital to know, in which area of application one is talking or reading. This must become clear in a document by the introduction. In a glossary the field of application must be indicated.

As an example, the term *character* has a completely different definition in ISO (the international standards body) and in a programming language or text processing application.

Problems with lack of terminology are numerous.

Developers very often need to describe something new and they 'just choose a name for it'. This leads to great problems in later stages of the product cycle.

A user is annoyed by variations of terms. As technology is perceived to be precise, different words let assume different meaning. A classic example is the use of the words End, Quit, Terminate, Exit to leave an application. If these variations are used, the user expects different behaviour of the program! A usual interpretation is, that End gracefully leaves the program with any necessary save, whereas the others abruptly quit the work...

When it comes to translation, lack of terminology becomes very expensive. The translator must identify and research words which might have specific meaning in this document. At least he will question their meaning and discuss it with you. Next time round you will have the same problem. In many cases you need to go back to the developers and ask them about this darn 'Graveyard' in the menu bar (the developer did not like the term 'Waste basket').

Terminology support marketing.

Terminology may also be a vehicle for marketing. Naming a product with a common term will change the meaning of the common term by the time to the product reference. Andreas Lötscher offers a rich set of such names in his book *Von Ajax bis Xerox.*

Terminology supports quality.

Quality assurance is a big issue nowadays. Without proper terminology no real quality can be achieved in documentation. Synonyms and undefined words are a source of errors and waste of money.

Both printed and on-line documentation should have a glossary defining the terms specific to the application.

The glossary in a document should be restricted to that document. However, it should be part of a complete terminology of the company.

Product names very often are taken from known concepts. It must always be clear whether a word is a product nameor has the generic meaning. If some component of the product is called Screen Processor, then the term screen must never be used alone, when the product component is meant. These definitions must be part of a company terminology.

Abbreviations are dangerous!

Abbreviations must be part of a terminology and glossary. There is only a limited number of short letter combinations, but a huge number of interpretations. It is by no means clear, what FTAM means in general.

The most critical things are abbreviations which are ‘invented on the spot’, because there was, for example, not enough space in a table. I have found such nice goodies as B.T. for Benutzertransaktion (user transaction).

Graphics are not that universal at all

After all of these problems with text to be translated you might think 'to escape' to safe ground with the universal language of graphics. Be warned. With the proliferation of icons and other graphics in software and documentation it became clear, that also graphics are sensitive to culture:

- Perception and interpretation highly depends on individual background (for example, education, religion)
- Positive feelings in one country about a graphic may be completely reversed in another.
- The implicit meaning or the emphatic function of colour differs significantly. For example, in Japan, mourning people are coloured white.

Graphics based on words and verbal analogies are not universal at all.

Verbal analogies use images of objects, which have the same name as a concept, for which the graphic should stand for. For example, The MS-Excel logo with combines X and L (pronounced ex-el). In German, this is pronounced eeks-el - which does not trigger the word Excel.

Using a crab-shaped sign for onkological department in a hospital is another example for this sort of graphics: The English word cancer both stands for the animal and the illness. This is not true in other languages (with some exceptions as in German).

Mythological and religious symbols are very dangerous.

Do not use an icon showing a person with black habit and ascythe on his shoulder (in our culture symbolising death) to illustrate fatal errors. What kind of interpretation can you expect from a Buddhist?

Associations with animals are also not universal.

To use the high reproduction rate of hares as a symbol for copy functions is at least questionable. In Australia these animals are pests! Dogs are pets to most Westerners, but food to people in the Far East.

Function of colours are subject to cognitive science - but cultures have distorted our visual system.

The warning function of yellow is unquestioned in Western countries, in China and Japan this colour expresses honour, but also childishness. Arabs associate Happiness and welfare with this colour.

Use colour in contexts, which do not allow symbolic interpretation.

Provincialism is found in many icons.

The common symbol for mail in most software shows the mailbox used in the outback of the US. On first sight I interpreted this as a dog's hut...

Similar problems arise with specific office equipment (nearly nobody knows a Rolodex in German speaking areas), clothes (a swim suite to indicate high temperature is not a good choice form many countries) or sports equipment.

Human figures are the most culture sensitive areas.

With the advent of fundamentalism, this area became even dangerous. Symbols for masculine and feminine are definitely different in many countries. Gestures showing hands with various positions of the fingers may produce a positive signal in one country and be offensive in another.

Cartoons, simple line graphics not indicating gender or colour are the best choice.

The more realistic the image is, the more it bears elements open to misinterpretation. Animation of object (distorted computer screen to indicate malfunction). With only few shapes it is possible to indicate danger or correct functions. It seems that the language of the face was distorted least by culture...

How to ease the translator's job

In most projects a translator is the only person aware of cultural differences between the developers and users of software. Hence the task of a translator often goes far beyond the translation of words. That's one of the reasons, why they earn some more money than a foreign student on assignment.

There will never be a complete set of rules about the culture in question, because it changes as well as our culture changes. These changes may even render existing documentation obsolete.

Without regard to any tool the work of a translator can be eased by observing simple rules.

- Define, whether to translate comments in example code or not.
- Provide a list of words, which are special in this application - a terminology. Provide references for these terms, as the environment may be essential for the translation. Indicate also probable use of synonyms, which must be avoided. If known, fill in the proper translations.
- Indicate different interpretations in rules and examples. For example, the 'tool' and the 'menu Tool' are not the same.
- Provide a list of all special words, such as keywords, names of macros, which must not be translated. This should be part of a terminology list.
- Always provide hardcopy (paper) to show arrangement of images, layout of the intended document and also changes by change bars. There may be subtle problems with code - not only of mathematical formulas.
- At best provide files in the format which the translator can handle directly. Not all translators use the same tools. Free lancers tend to stick to known tools to avoid surprises, rather than experiment with new tools. This may request to provide DOS files rather than MS-Windows files. Assure correct code page!
- Always provide complete documents, even if the translator needs to work only on part of it. For proper translation the environment is needed.
- Indicate changes from the old version to the current. This helps greatly to find the proper locations. Even if the translator uses specialised tools, this provides feedback.
- Keep formatting to a minimum. Even when using a text processing tool, the translation work is eased, if bitmap graphics are absent, multi-column features are avoided etc.
- When providing unformatted files, do not hyphenate. To call translation tools the complete words are needed.

Evolving tools help the translator to keep up with new versions of software.

Much documentation is a new version of existing one to cope with software updates. It is absolutely boring to find out, that only few things must be changed on some pages, and it is also a waste of money to send completely new files to the translator.

Terminology data bases are a treasure chest of professional translators. According to many of them, there is little exchange in this field. If you are working in long term contract, ensure that the terminology data base is your property. This helps in continuity and eases transition to another service provider.

MultiTerm is a product of Trados GmbH in Stuttgart. It is pre-loaded with a GUI terminology in some languages.

An in-house terminology data base would ease the work of writers. It turns out, that only very large companies are aware of the value of such a data base. Medium sized and small companies do not have a post to maintain this device. Hence the writers are in constant fight with developers who invent new words for not so new things.

Tools for a translator such as Translation Manager or Translator's Workbench help a great deal to keep track with frequently changing documents. The changes in documentation from one software release to the next may be less than 10%, but wide spread. These tools keep references from a source document to the target document. Using a very intelligent pattern matching process they pinpoint changes and translate already translated passages without human intervention.

Translation Manager is a product from IBM running in OS/2.

Translation Manager is aware of DCF tagging and hence be used also to translate dialogue elements for the Presentation Manager - if they use the dialogue Tag Language. It is not known to me, whether dictionaries of specific application fields are available. IBM recently implemented a number of enhancements, which are solutions to problems reported at a former SHARE Europe Conference by Bernard Chombart.

Translator's Workbench II is a product from Trados GmbH in Stuttgart.

This tool has functions nearly identical to Translation Manager. I have not heard about this product since end of 1992.

A checklist for software

Do not have any text (command names, messages, menu items etc.) hard coded in the program.

Interface design

- Do not fix the size of a message display.
- Allow much space for translation of labels for fields or buttons.
- Take default values from a table or initialisation file, where they can be translated.
- Use templates in an initialisation file for items such as phone-numbers, currency-format, date format (if the operating system does not provide international services).

Messages

- Do not use combined strings to form a message.
- Include comments in message file.
- Never use substitutions instead of alternative messages.

Commands

- Put commands, macro names, synonyms and abbreviations in a table.
- Check user responses against a table.

Data

- Do not assume the code page of the data.
- Do not assume length for data like date, time, phone number, ZIP codes etc.
- Allow the use of national characters, do not restrict the character set.

Icons

- Show hands only in action, not in gestures
- Avoid feminine or masculine accessories (jewellery)
- Always show the right hand
- Do not make metaphors based on animals, colours or people
- Check for cultural bias in the destination country

Literature

- Apple Programmer's and Developer's Association: *Human Interface Guidelines*. Form APDA # KNBHIG. 1986.
- Bouch, Debb: *Adding National Language Support to Applications*. Proceedings of SEAS AM 1987.
- Daube, Klaus: *Text and Code - A Dragons Pond*. Proceedings of the 30. G.U.I.D.E. conference in Basel. G.U.I.D.E. Headquarters, Ebikon, 1989.
- Daube, Klaus et al: *National Language Architecture*. SHARE Europe White Paper. Geneva, 1990.
- IBM National Language Technical Centre: *National Language Design Guide volume 1, Designing Enabled Products*. Janauary 1991. Form number SE09-8001.
- IBM National Language Technical Centre: *National Language Design Guide volume 2, National language Support*, Reference manual. 1992. Form number SE09-8002.
- IBM National language Technical Centre: *Keys to Sort and Search for Culturally Expected Results*. February 1990. Form number GG24-3516.