# Proposal for a prototype integrated system for non-programmer's personal and professional information management (PIM) in natural language

THE-RC 50933



Eindhoven University of Technology
Computing Centre Note 12

# PROPOSAL FOR A PROTOTYPE INTEGRATED SYSTEM FOR NON-PROGRAMMER'S PERSONAL AND PROFESSIONAL INFORMATION MANAGEMENT (PIM) IN NATURAL LANGUAGE

ir. Jan Hajek

December 1982

Table of contents:

Abstract

Proposed are applicative and qualitative requirements and a development schedule for a personal and professional information management system. Special attention is payed to user-friendliness, simplicity and economy. The system is intended for daily use by white-collar workers who are non-programmers.

Keywords: documentation, retrieval, free-text, natural language.

## I. The Scene.

Recent techno-economical advances in computing and communications (= compunications [1]) offer a promise to help us to cope with at least the following plagues of our time- and cost-conscious (post) industrial society:

- data inflation and information explosion;
- bureaucratization i.e. the ever increasing volumes of paper to be typed, copied, mailed, read, filled-in, filed, searched, updated, reorganized, guarded and finally disposed off;
- complexity and frequency of decision-making (personal and professional evaluating, comparing, weighting, asking yourself: who, what, which, when, why ... how, how much?).

Major obstacles to our progress are:

- high costs of labour in general and of the qualified labour in particular;
- lack and high costs of compunications know-how, further amplified by computer-shyness;
- mediocre quality and high costs of current "solutions" to the above mentioned plagues; typical are "six million dollar solutions" which must be operated and maintained by "crews of MIT graduates";
- inadequate and clumsy means of communication (man-man and man-machine).

## II. The Scenario.

Our goal is to remove or at least to reduce (the impact of) the obstacles to progress. One weapon in our struggle to remove bore and chore and drag and lag from our offices and workrooms is automation of personal and professional paperwork.
Office automation [2] is not limited to, yet rests safely on, three cornerstones:

- easy and integrated communications (interpersonal and interorganizational; by voice, messages and images);
- semi-automated text manipulation (entry, editing of inputs, formatting of outputs in what is called word-, text-, document-processing);
- information management.

Commercially available and more or less affordable are systems for communications and text manipulation which just started to proliferate, although not yet in integrated forms. However recently announced joint ventures (Xerox and DEC and Intel) clearly indicate that "major integrations are in making" (both in technological and corporate sense).
Aside from business reasons, it seems obvious that the communications plus text manipulation are of more "basic nature" and therefore less application dependent than information management.
Flexible, user-friendly and affordable solutions for info-management are neither available nor they can be expected in near future for at least three reasons.
First it is a lucrative business to (re)sell (semi)taylored, customized solutions. Secondly it is not that easy to do such a job properly, because number of factors enter the design and enforce heavy commercial versus engineering trade-offs. It is not trivial to keep the design balanced under such circumstances. The third reason is the sad fact that the designers are often not more than enthousiastic "hobbyists" without any extensive, practical experience in this application domain (compare with the lifetime experience in [3]).

Disbelieving Thomases are reminded of an analogy concerning long ago standardized "codes for information interchange" (USASCII and EBCDIC) versus totally unstandard "job control languages" for computer "work flow management". Yet another example are well-standardized low-level data transmission protocols versus a "wild bunch" of higher-level communication, command and control (C3) protocols. These examples illustrate that "the higher the level, the less standardization" is available on the market.
While it is proper to consider advanced communication networks (local and long-haul) as "electronic infrastructures" or "communication infrastructures" of our society, it is also proper to consider information management tools as "management infrastructure" of our offices, workrooms and homes of near future.
Our PIM could become a tool to be used daily by white-collar workers: secretaries, clerks, bosses, technicians, laborants, librarians, managers, assistants, professors, etc.
PIM is intended to be one of the first building blocks of the information infrastructure of our University and of our homes.
In order to be useful, it must be integrated. Hence a glue of commercial mini-packets would be more of a problem instead of being a solution.
The idea of an integrated PIM was born out of private discussions between dr. Vlado Stibic and the present author in late 1980.
Later (after the draft of this report was typed) we were pleased to see our ideas confirmed by the famous (and expensive) professional consultants [4]:
GEBRUIKER BEWANDELT VERKEERDE WEG NAAR TOEKOMSTKANTOOR:
"BETER DEEL VAN MANAGEMENTWERK AUTOMATISEREN":
Het adviesbureau Booz, Allen & Hamilton is na een studie bij vijftien concerns, waaronder AT&T, Burroughs, Control Data, IBM en Xerox, tot de conclusie gekomen, dat de nu ingeslagen wegen naar het zogenoemde kantoor van de toekomst niet helemaal de juiste zijn. In plaats van ons te richten op de automatisering van administratief werk, zo stellen de onderzoekers, zouden we veel meer de elektronische hulpmiddelen ter beschikking van het management en andere professionals moeten stellen.

III. Applicative Requirements for PIM, ordered by priority:

0. Personal word processing is assumed to be commercially available at least as a screen-editor.
1. Utilities (= basic functions e.g. file creation, update, etc.).
2. Personal and professional documentation:
2.1. Retrieval (= search, checking, inquiries, queries) as specified in Appendix 2 and in [3].
2.2. Indexes like e.g. an ALPHAbetical index of a certain item (= field of a record), index of CONCORDances, KWIC/KWOC i.e. Keywords In/Out of Context.
3. Graphical presentation (= simple charts, diagrams, statistics).
4. Intelligence amplifier (= deductive Question-Answerer).

Examples of useful work-flows:

(file and query) → RETRIEVE → all records matching the query

file → KWIC → all keywords → USER → selected keywords → USER → (query and file) → RETRIEVE → matching records

(file and query=[ ) → CONCORD → all places of referencing

(file and query) → CONCORD → all places of use of keywords (e.g. consistency test)

The unique property of PIM will be the mutual compatibility of all applicative functions so that data will be able to flow freely from one functional module to another. We can envision the cooperation among modules to proceed through standard interfaces (i.e. record and message formats) and data pipelines (I/O/I/O). From such compatible and complementary data-flows will result synergistic effects and applications. This integrated approach is completely different from the traditional way of glueing semi-compatible programs together.

We consider the RETRIEVAL as the most complicated and the most useful basic and applicative function in PIM. Unlike in most of the commercially available retrievals PIM's retrieval will be well parametrized [5]:

- default mix (= default procedures and default layout);
- typical mixes (= several useful combinations of parameters will be triggered by a single generic command);
- individualized mix of commanded parameters.

The RETRIEVAL is globally specified in Appendix 2.

The function of a retrieval is to provide a "high-level context-addressable associative memory" for users. Retrieval serves as a non-deductive Question-Answerer (Q/A-function) which utilizes only the factual data (= texts and numbers) stored in files. If feasible, a smart retrieval might be designed and implemented later. "Smart" would mean certain capabilities to deduct new facts from stored facts.

In any case a retrieval is users' "memory gap filler". Therefore a retrieval is a key function of any Management Information System (MIS) and it must form the nucleus of our Personal and Professional Information Management System PIM.

Basic Functions

Upon user's command various applicative and basic functions (= operators) will be evoked and executed on data (= operands). For a list of PIM's basic functions see the sample dialogue between a user (U:) and computer (C:) program PIM. Comments follow the % until the end of line.

Example of a dialog:

```
% Power is ON and PIM is loaded (= resident in computer's
% memory).
C: PIM is ready to serve, enter your command followed by RETURN-
   key.
U: ?  % User pressed ?-key or some, if any, other key(s) thus
      % entering an illegal command, e.g. an empty line
        (properly % followed by RETURN).
      % If user entered a legal command then PIM will not
```

```
        % present its MENU of commands. PIM's behaviourial model is
        % a restaurant where a gourmet (who knows what he wants and
        % what he can get) will order directly without looking into
        % a MENU.
C: Choose from PIM's MENU of functions (capital letters denote
   synonyma i.e. equivalent words and abreviations of at least
   3 letters are allowed):
C: ? = SOS = HELP requested by user.
C: 0 = UNDO = REVERSE effects of last command executed.
C: 1 = DEFINE = DECLARE data or file.
C: 2 = RETRIEVE = RETRIEVAL = QUERY = INQUIRY searches i.e.
       tests i.e. checks which records match query conditions.
C: 3 = KWIC lists Key-Words-In-Context index.
C: 4 = ALPHA = alphabetical index (specify item).
C: 5 = CONCORD = concordances.
C: 6 = GET = ENTER = INPUT = READ data from Display, File.
C: 7 = PUT = EXIT = OUTPUT = WRITE data to Display, File,
       Printer.
C: 8 = UPDATE = CHANGE = MODIFY = ALTER data.
C: 9 = INSERT i.e. place new data in between other data.
C: A = COPY = MOVE = ASSIGN data to destination from source.
C: B = DELETE = REMOVE data which match a condition.
C: C = SORT file into new file (non-ascending/non-descending).
C: D = MERGE two files into new file.
C: E = TYPO = SPELL searches for possible typing/spelling errors
       in text.
C: F = INFO provides information about PIM and its ARCHIVE
       (= database).
C: G = GRAPH = DRAW = CHART = SHOW a diagram.
C: H = ACK = ACKNOWLEDGE = ACCEPT defaults suggested by PIM.
C: I = NAK = NACK = REJECT defaults suggested by PIM.
C: J = MENU of commands available to the user in current
       situation.
```

C: K = KILL = STOP execution of a computation process (PIM's module).

U: 2 % or RET or RETR or RETRI or RETRIE or RETRIEV or % RETRIEVE or RETRIEVA or RETRIEVAL.

C: FILENAME = ?

U: LIBRARY % or other existing filename.

C: QUERIES = ?

U: DATE > 1979 and DATE < 1982 and AUTHOR = STIBIC / AUTHOR = HAJEK ??

C: Nr. of records matching the queries = 69:

C: 1. HAJEK J, PROPOSAL FOR A PROTOTYPE INTEGRATED SYSTEM FOR NON-PROGRAMMER'S PERSONAL AND PROFESSIONAL INFORMATION MANAGEMENT (PIM) IN NATURAL LANGUAGE, Date = ..., ..., etc.

2. .... etc.

## IV. Qualitative Requirements for PIM.

Qualitative goals ordered by priority:

1. Friendliness = ease of interactive use for non-programmers.
2. Universality = functional flexibility and compatibility and standardization.
3. Effectivity = usefulness and good performance (short response time).
4. Economy = cost-effectiveness in terms of investment in equipment, development, use, maintenance, transfer.
5. Reliability = robustness and correctness.
6. Safety = protection (= security) and insensivity.
7. Intelligence = smartness.

Note that there is an interdependency (often of an implicative kind) among these goals. Thus a system with poor response time or an unreliable system is certainly not a friendly one. But a reliable and fast system might not yet be a friendly one. In other words, reliability and good performance are necessary but not sufficient conditions for friendliness.

Qualitative subgoals

So noble and different goals need not become too contradictary. We have found that they have one Greatest Common Denominator, namely the SIMPLICITY. Simplicity supports most of the goals except for "smartness" and in some, but not all, respects the performance. Hence the general trade-off formula for PIM should be:

80% of SIMPLICITY + 20% of SOPHISTICATION.

In order to prevent being accused from sounding a lot of platitudes and/or wishful thinking, we specify the way to simplicity by the subsubgoal formula:

SIMPLICITY = UNIFORMITY + SERIALITY.

Still sounds vague? Well, e.g. a uniform structure of records and messages and limitation to very few (2 or 3) "types" of items (= variables) means not only saving man-hours and computer resources but it means also COMPATIBILITY, EXPENDABILITY, TRANSFERABILITY and more. All these positive properties are further supported by SERIALITY in data storage and processing. The advantages of the SERIAL approach are multifold:

- simplicity of programming hence low cost and high reliability;
- high raw speed and flexibility due to the following facts:
- Linear string search is most flexible, avoids costly (in space and CPU-time and programming) buildup, storage and updating of extensive, multi-language dictionaries and huge pointer-arrays. Moreover there exist superfast, multi-key string search algorithms with sublinear performance. The critical search loop can be written in 3 or 4 machine instructions. Such a critical loop and/or disk-reads will dominate PIM's performance.
- Sequential, block-wise disk-reads save a lot of head-moves, track-switching and rotations (and fault-paging i.e. swapping in virtual memory systems).

Data Structures

User may define, store and access data (= operands) which are grouped/fragmented as follows.

Archive is a collection of files. Archive serves as user's data base.

File is an (un)ordered collection of homogenous records. File is identified by its filename.

Record is an ordered set of items (= variables). Record is implicitely identified by its recnr (= record number).

Item is an elementary unit of data. Item is identified by its tag, which is a name (= identifier) of an item abreviated to few (4-6) characters.

Item occupies a contiguous storage space (= field) in a record. Item is characterized by its type, which may be either:

T-type for texts i.e. "left-justified, left-compared" strings of characters (= alphanumeric symbols) or

N-type for numbers which may be (un)signed integer numbers and real numbers in fixed-point or floating-point notation.

These two types would be sufficient if all numbers (N-type) will be stored as a "right-justified, right-compared" alphanumerical string. If for pragmatic reasons such a textual representation of numbers would not appear suitable, then N-type will have to be replaced by two more specialized types:

R-type for real numbers in floating point representation;

I-type for integer numbers.

The black and white reasoning enforced upon us by Boolean logic is not sufficient to capture the complexities of the real world inhabitated by real people (= non-programmers!!!). Therefore logical values will be represented as one-character strings:

Y(es), N(o) and ? (= unknown, undefined, not announced, uncertain).

Hence PIM will employ ternary fuzzy logic (true, false, trulse i.e. fuzzy) as specified in Appendix 1.

An item has always a value (either entered i.e. commanded by a user or assigned by PIM post-initially i.e. per default).

Special kinds of value: undefined value, empty value, minimal value, maximal value.

Pragmatics: non-alphanumerical representation of numbers (R-type or I-type) means fixed-sized format (= number occupies fixed-sized field in a record), while strings (T-type) have variable size representation. Hence for strings it is important to know the maximal size of a string (e.g. 65000 characters) and also minimal size, which for an empty string is zero i.e. not a single character. Caution: space i.e. blank is also a character, thus a string of one or one or more blanks is a non-empty string.

For non-programmers it may be less confusing not to distinguish between an undefined value and an empty value. Thus a yet undefined string could have an empty value represented as a string size equal to zero. For numbers we will need a special non-zero value for undefined number. This will prevent us from confusing e.g. an unknown (because not announced) price with a "give-away" price of $ 0.00. An implementation hint: use the type-field to indicate the undefined and/or empty value.
Thus another thumbrule for PIM is:
KIS = KEEP IT SIMPLE (but not oversimplified).

Detailed qualitative requirements for PIM

Detailed requirements further specify and illustrate what kind of behaviour we expect from PIM.

1. Friendliness-requirements
   Definition: Friendly = interactive and simple and concise and readable and understandable and helpful and self-instructive and smart and robust and fast.

1.0. Good (error) reporting in terms intelligible to non-programmers in their native language (Dutch, English, etc).

1.1. Prompting = PIM takes initiative to ask the user what PIM needs (data or (sub)command) in order to execute user's wish.
Caution: avoid unnecessary chatter; avoid overly fragmentized dialog. Allow the user to supply at once (= in a single command) all what PIM needs. Only if any of the required items have not been entered then PIM will prompt the user for remaining items.
This method allows for both the experienced user and the beginner. Thus an instant mutual adaptation between the user and PIM is realized.
If a user will leave some value unassigned then PIM will:
- mention this to the user and
- if possible then PIM will suggest a default value.

Thereafter a user will be free to:

- accept the suggested default or
- assign a value, which may be an explicitely commanded undefined value.

Unless suppressed, defaults may be reported to the user who will have to either acknowledge (ACK) or not acknowledge (NAK) these default assignments (see the MENU above).

1.2. Menu-driven = multiple choice of functional commands.
Menu is a list of commands (possibly applicable in a current situation) with short explanations (see above the example of a dialogue).

1.3. Help from PIM to the user:

a. Upon user's request.

b. After user's mistake.

c. After too many user's mistakes PIM will switch to the more redundant (talkative) TEACH-mode of dialogue.

d. After a period of flawless communication (e.g. 5 legal commands) PIM will:

d1. If PIM was in TEACH-mode then PIM will switch back to NORMAL-mode.

d2. If PIM was in NORMAL-mode then PIM will suggest to the user to try a more concise form of commands. If this proposal is rejected by the user, then PIM will not ask anymore within the current session.

1.4. Minimum of syntax to learn e.g. only one level nesting in queries will be implicitely determined by binding powers of logical (= Boolean) operators (and = &, or = /, not = -) and relational operators (=, <, >, <=, >=, <>).

1.5. Implicit (= default) truncation in retrieved keywords (but not in keyphrases). Thus a user will just use the root (= stem) of a keyword to find all its reincarnations in a file.

Such a default truncation takes care of:

- prefixes (e.g. in English: anti-, be-, de-, for-, in-, mis-, pre-, re-, un-, etc.),
- infixes (e.g. hyphenation -, slashing /, personal names without knowledge of exact spelling like e.g. HA?EK, elimination of differences in representation (= notational differences) of codes like e.g. 1,000.000 vs 1000000, and other pattern matchings in texts),
- suffixes (e.g. -able, -ally, -ant, -ation, -ations, -ed, -er, -es, -ess, -est, -ies, -ility, -ing, -ings, -ion, -ity, -ities, -ization, -less, -ment, -ness, -s, -s', -'s, etc.)

in most European languages "automatically" (thanks to the natural intelligence of the user) without using special markers.

Only if necessary a marker (e.g. _) will be used to suppress the default truncation at either or both ends of a keyword. This is needed mainly for very short roots so that a query like _CAR_ / _CARS_ will prevent a pseudo-match with e.g. CARPENTER, VICAR, CARE, CARBON, CARL, PRECARIOUS, CARSON.

The best will be to implement a pattern matching more general than just "truncations and one-letter-infixes". A pattern matching will allow to retrieve keyphrases (= strings) with unknown parts (= substrings) of variable size like e.g. _HA??EK_, _STIBI??_, etc.

Until now the common practice is:

- either a use of huge dictionaries which are natural language dependent,
- or a natural language dependent preprocessing of text files and queries (eventually combined with still very large dictionary),
- or extensive explicit use of special markers in queries.

Multi-word keyphrases will be interpreted:

- As a single long keyword (= keystring).
- Optionally as a concatenation of truncated roots.

1.6. Implicit (= default) normalization of text inputs (data and queries). A proper mini-normalization is compression of more adjancent blanks (= spaces) to a single one. This is a necessity for more successful searches and sometimes it will even improve layouts and save the space in storage and output media. Further possibility is to remove all adjancent blanks between a letter and word-separators like .,:; etc.
Note that too much normalization might harm. Normalization can be performed:
- (per default) upon entry (= input) of data and query,
- (optionally) upon each read from the secondary store (= disk) so that the original formatting will be preserved.

1.7. Dictionary of synonyma should be available.
Initially a small (natural language dependent) dictionary should contain some:
- general synonyma (e.g. colour = color, modelling = modeling, US = USA = U.S. = United States = United States of America) and
- specialistic synonyma (e.g. I/O = I-O = input/output).

Such a synonyma matching capability is easily implemented even if a retrieval based on sequential search (= dictionaryless because no inverted file) is used as we suggest. It is just necessary to preprocess user's query via a small dictionary of synonyma and then to search for them too by means of our multi-query-per-record (saves disk reads) serial retrieval (see Appendix 2).

2. Universality-requirements.
Definition: Universal = application-independent and unrestricted (i.e. flexible, variable, redefinable).
User will define (= declare) his file of (logical = user's) records by entering the command DEFINE and further:

- filename (e.g. LIBRARY).
- list of items' attributes:
  - tag = abreviated name of an item,
  - type = kind of item.

That's all. Optionally a user might define his own (= specific to a particular file) maximum size allowed for a text (T-type) item.

Once the file is defined PIM will remember:
- number of items in a record (will be fixed per file),
- tags,
- types,
- sizes of items (if a maximum size is specified).

All this information will be used:
- to build records,
- to check on legality of entered data items.

3. Effectivity-requirements.

Definition: Effective = useful and fast (= low response time). A slow system is useless. An average response time for an average query (of 5 keywords) on an average personal file (of 300000 characters i.e. either 1000 records of 300 characters or 1500 records of 200 characters) should be few (dozens) of seconds on larger (mini)mainframes or should be below three minutes on lowest priced personal microcomputers.

As far as the usefulness concerns some "documentation freak" might say that his hand can beat any computer. Maybe, but people are no documenation specialists, they do not know how to organize their data for optimal searching and they do not have an iron will to maintain a rigid discipline in their paper-files. They also do not have to. The brute force of PIM's serial search will find whatever a weak human flesh has burried in, and only fragmentarily remembered from, a mess of free-text.

In any case the usefulness will have to be tested in the field by unbiased users (= non-programmers).

4. Economy-requirements.

Definition: Economic = cost-effective in terms of resources spent.

A personal microcomputer which could run Pico-PIM (say 1000 records of 300 characters = 300000 characters in ARCHIVE = two mini-floppies).

Micro-PIM could run on a more expensive semi-professional office microcomputer. Further upgraded versions for mainframes of various power could be Mini-PIM, Midi-PIM and Maxi-PIM.

Investment in man-hours spent on development, use, maintenance, extension/modification and transfer will be minimal due to the fact that our major weapon in our struggle for economic solutions will be the SIMPLICITY through SERIALITY.

5. Reliability-requirements.

Definition: Reliable = robust and correct.

We do not expect that people's lives and health will directly depend on PIM (unless used in pharmacies!). Hence no special precautions except for systematic testing will be taken to assure its correctness.

Much more care will have to be paid to making PIM robust. Defensive programming i.e. extensive and intensive checking of all inputs and outputs will be performed by means of:

- I/O-assertions (= entry/exit checks of conditions),
- cross-checks,
- plausibility-checks (if defined by a user).

6. Safety-requirements.

Definition: Safety = protection (for security) and robustness (certain insensivity to mistakes).

Considerable protection would be obtained by allowing a user to (re)define a coding scheme for his texts (T-typed items). A simple, personalized, hence secret, coding scheme, e.g. derived from frequently occurring bigrams (i.e. pairs of characters appearing in a text), would be natural language dependent and would have two advantages:

- protection,
- text compression (which has several advantages:
  - saves up to 50% of the valuable disk storage space;
  - disk I/O transfer speed could almost double);
  - more records stored per track hence less paging;
  - less head moves per record, hence faster.

Although easy nuts to crack for National Security Agency's cryogenic and opto-electronic codebreakers (NSA's budget is 10 times that of the CIA), an occasional peeker or weekend burglar would still break out his teeth on these bites.
Note: While coding is performed only once (upon entry), decoding will have to be repeated upon most of internal record-reads from disk. Decoding will be necessary in order to allow comparisons, sorting, searching, etc. Hence the decoding must be fast in order not to degrade the performance seriously. It would be nice if our hardware guys could build a microprocessor based (de)compressor. It would be placed on the disk I/O channel. The user-interface would have to appear in a programming language as two I/O statements: WRITECOMPRESSED and READCOMPRESSED plus a CODEKEY(para) command for personal parametrization of the (de)coder/(de)compressor.

7. Intelligence-requirements.

Definition: Intelligence = ability to accept inputs in restricted natural language [7] and certain ability to deduce new facts from old ones.

By 1980 deductive Question-Answerers are no more a science-fiction.
The thumbrule for "smartness-as-perceived-by-users" should be:
VISIBLE SIMPLICITY through INVISIBLE SOPHISTICATION.

5. Schedule of development for PIM.

We recommend staged development in five phases where man-months are based on 40 hours work-weeks and average industrial working experience with hardware and software involved and support typical in the industry.

1. Proto-PIM = Prototype Version of PIM will be simple yet well usable i.e. not oversimplified. Here simple means:
   - Only the RETRIEVAL and few indispensible basic functions like file creation, input and output of data, etc. will be implemented.
   - No fuzzy logic yet.
   - One useful mix of defaults plus a limited choice of commands.

   We are convinced that we should start to develop PIM neither top-down, nor bottom-up but rather mid-out i.e. from its nucleus (which is I/O + RETRIEVAL), thus being forced to resolve complicated design issues, clashes and trade-offs in their mutual interplay. A "fundamentalistic" start concentrating first on elementary functions "in their splendid separation" would lead to painful discrepancies later. PIM must become a viable workhorse, not a laboratory vegetable.

   Rapid prototyping and deployment of several PIM-stations is recommended. We consider it essential to collect field-data on users' experience, impressions, wishes and suggestions. Hands-on experience by unbiased users is better than any kind of educated guesswork on what "friendly and easy" means to non-programmers. If feasible, Proto-PIM could be implemented on a stand-alone, low-priced personal microcomputer (= Pico-PIM).

   Work-investment is estimated at 6 to 9 man-months.
2. Full-PIM = Full-fledged Version of PIM with all basic and applicative functions. An implementation on more powerful (semi)professional office microcomputer might be appropriate (= Micro-PIM). Work-investment is estimated at 9 to 12 man-months.
3. Smart-PIM = Intelligent Version of PIM capable to accept inputs in restricted natural language and with certain deductive abilities.

   Work-investment is estimated at 9 to 24 man-months.

4. Net-PIM = Network Version of PIM for distributed office environments connected via local network. Front-end interface would have to be added plus few commands for remote communication (MAILTO, MAILFROM at least).
   Work-investment is estimated at 6 to 9 man-months depending on services (not)embedded in a local network.

Note that this proposal has resulted from the earlier draft THE-RC 43517, February 20, 1981, where we have perhaps over-praised the virtues of personal computer for this porpose.

Acknowledgement is due to dr. Vlado Stibic whose lifetime experience with documentation and retrieval cannot be over-estimated. This report is based on conversations with Vlado Stibic, who was awarded with Kluwerprijs for his book [3]. Encouragement received from my boss drs. Ben Morselt, Director of the Computing Centre of Eindhoven University of Technology, is also greatly appreciated.

References

[1] OETTINGER, A.G.
Information Resources: Knowledge and Power in the 21st Century, Science, vol. 209, 4 July 1980, pp. 191-198
Professor Anthony G. Oettinger is chairman of the Program on Information Resources Policy, Harvard University, Cambridge, Massachusetts 02138, USA

[2] MORGAN, H.L.
Research and Practice in Office Automation, Proceedings of IFIP Congress, 1980, Tokyo, Japan, pp. 783-789
Dr. Howard Lee Morgan is professor of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104, USA ((215)243-7731)

[3] STIBIC, V.
Personal Documentation for Professionals - Means and Methods, North-Holland, 1980, ISBN 0-444-85480-0
Awarded with Kluwerprijs 1982
Dr. Vlado Stibic was until October 1980 with Philips Gloeilampen N.V., Information Systems and Automation (ISA-R), Eindhoven, The Netherlands

[4] Computable, February 13, 1981, p. 2 (in Dutch)

[5] STIBIC, V.
A few practical remarks on the user-friendliness of on-line systems, 1980, 12, J. Inf. Sci., 2(6) 277-283

[6] GEBHARDT, F., STELLMACHER, I.
Design criteria for documentation retrieval languages, 1978, J. Amer. Soc. Inf. Sci., 29(4) 191-199

[7] STIBIC, V.
Tools of Mind - Techniques and Methods for Intellectual Work, 1982, ISBN 0-444-86444-X, North Holland Publishers

[8] GROVES, L.J.
Using Simple English Sentences to Call Procedures, SIGPLAN-ACM, vol. 17, no. 11, November 1982, p. 31.

## Appendix 1: Fuzzy logic

The evaluation of the following statement

B := FUZZ(A[0] & A[1] & A[2] & ... & A[i] & ... & A[n])

is performed according to the following operational definition of FuzzyAND (ANDF):

ifexists i sothat A[i] = false then B := false else
ifexists i sothat A[i] = fuzzy then B := fuzzy else B := true.

The evaluation of the following statement

B := FUZZ(A[0] / A[1] / A[2] / ... / A[i] / ... / A[n])

is performed according to the following operational definition of FuzzyOR (ORF):

ifexists i sothat A[i] = true then B := true else
ifexists i sothat A[i] = fuzzy then B := fuzzy else B := false.

Fast conditional evaluation of ANDF and ORF is possible (in analogy to conditionally evaluated CAND and COR).

## Appendix 2: Algorithmic definition of RETRIEVAL

```
openfile(filename);
record := readnextrecordof(filename);
while record ≠ endoffile
do openq(listofqueries);
   query := readnextqueryfrom(listofqueries);
   while query ≠ endofqueries
   do if matches(record with query)
         then write(record to medium)
      fi;
      query := readnextqueryfrom(listofqueries);
   od;
   closeq(listofqueries);
   record := readnextrecordof(filename);
od;
closefile(filename);
```