USER INTERFACE AND DIGITAL REMOTE SENSING ANALYSIS: Reconciling Program Development, Applications Research, Data Production and User Interface During System Design.

Berne Grush
PERCEPTRON Computing Inc.
4800 Dufferin Street
Toronto, Ontario M3H 5S8
Canada

Proceedings:
Commission II Symposium for
the XVth ISPRS Congress
Rio de Janerio
June 1984

ABSTRACT

Four hypothetical classes of digital image processing systems for remote sensing data production and analysis are described in conjuction with problems typically encountered in their software design, especially as this applies to the user interface.

A short discussion of software system design considerations, again with special reference to the user interface, is followed by a broad-spectrum definition of the 'compleat' user interface.

Finally, a short dissertation on PERCEPTRON'S machine independent, all-software entry, EASI/PACE.

Introduction

For purposes of this discussion, I will hypothesize four classes of digital image processing systems, two at production level and two at applications level. The systems that each of us may be familiar with probably do not fit neatly into this scheme, but so it is with all forms of classification from the biological to the unsupervised.

I will describe what I perceive to be the state of the user interface in many of these systems -- or at least the ones I happen to be familiar with. With respect to any criticisms I may make, please keep in mind that good programmers can make machines do virtually anything, but that few of them have been exposed to the full range of design considerations for the user interface.

Indeed, I have read descriptions of what appear to be very good interfacing techniques, and I think that several groups are, by now, paying this problem the attention it deserves. In the Hofman et al 1983 ERIM paper outlining the MIDAS project, the Transportable Applications Executive (TAE) is described. This a common user interface providing menuing, on-line help and a command language with parameter passing and branching. Elsewhere in this same paper Hofman et al write:

"A critical component of any software package is that part with which the user communicates with the program. Many users will judge the usefulness of a package more by its user interface than by the functions it supplies. If the man/machine interface is awkward and difficult to use, the user will be less likely to invest the time and energy it takes to learn to use it. A good interface allows the user to concentrate on solving his application problem, rather than solving the problem of how to use the software."

[Hofman, L. B., Erickson, W. K., Donovan, W. E. MIDAS: A Microcomputer-based Image Display and Analysis System with Full LANDSAT Frame Processing Capabilities. Proceedings of the Seventeenth International Symposium on Remote Sensing of the Environment. p. 170.]

Following the discussion of current user interfaces, I will present a description of what I think should comprise the complete user interface.

My parting discussion will be a description of a system that I have been closely involved with for the past few years. Although this system does not yet cover the full range of hypothetical systems, its structure has the potential to do so. The user interface inherent in this system will allow more analysts to enjoy a more direct relationship with remotely sensed imagery.

1: First Level Production Systems (P1)

The first class of image procesing systems for remotely-sensed data involves preparation of basic end-user products directly from HDTs or from CCTs on which minimal preprocessing has already been performed. These products require computation of various degrees of partial or complete geometric and radiometric correction for CCT or hardcopy output. P1 systems enjoy user acceptance in direct proportion to their throughput and hardcopy quality. Speed and throughput requirements are generally detrimental to system flexibility, in particular where array processors are employed.

These systems are constructed and operated by computer programmers. P1 user interfaces usually reflects this fact, but this is seldom a critical factor at this level.

2: Second Level Production Systems (P2)

Often further correction, enhancement, registration, mosaiking and perhaps some filtering are required before a

product, suitable for cartographers and analysts, can be created. Again these products are in the form of CCT or hardcopy. Systems supporting this processing must be more flexible than P1 systems and typically meet end-user expectations less often than P1 systems do. The trade-off between flexibility and speed (general purpose computers vs array processors) may be misunderstood or ignored, in favor of the tendency to select hardware without regard to software.

The user interface starts to become important in P2 systems since some computer-literate analysts may like to have a hands-on relationship with the data. Unfortunately, this typically takes the form of peering over an operator's shoulder.

3: First Level Application Systems (A1)

Usually billed as 'turnkey', end-user stations, first level application systems provide more specialized enhancement, arithmetic (ratio, add, difference, principal components, etc) and classification capabilities. User controllable image raster display hardware is a system prerequisite. System output tends to consist mainly of specialized hardcopy and statistical tables, charts, plots and matrices. In A1 systems, flexibility is crucial. Various combinations of operations may have to be constructed for each end-user's needs. Manual, spatial input plays as important a role as does data quality and system throughput. Manipulating the processing programs can become more important, and more difficult, than manipulating the data - from both the programmer's and the user/analyst's points-of-view.

The user interface is absolutely critical at the A1 level. Many computer literate analysts, eager to get hands-on control of data processing, are discouraged by intimidating and humiliating machine rebuffs. They return to "over the operator's shoulders" mode. Many A1 systems that are available either fall into disuse or require far too many hours of operator coaxing to achieve desired results. I have been told on several occasions that: "the results were not exactly what we wanted, but it was too much trouble to re-do the work." In fact, I am familiar with a system whose documentation was so troublesome to read that one of the chief analysts using the system was unaware that the system incorporated classification programs, even though these algorithms were of central importance to the system. He hired a programmer to write the apparently 'missing' programs.

4: Second Level Application Systems (A2)

Second level application systems provide grist for the paper mill, more of which should be carried out, considering our relative innocence in employing digital image analysis techniques on remotely-sensed imagery. Here is where highly computer-literate analysts wish to specify new algorithms for rectification, enhancement and especially for classification, as well as improving existing algorithms. Programmers are hired to design, code, test, debug and struggle with mysterious things such as FORTRAN, JCL, RSX-11M, TASKLINKER, VMS, DMAs and system crashes. Who has time to consider mundane things such as the user interface, when one can so directly challenge the MACHINE?

The A2 user (programmer) interface? Things are looking even worse. There isn't one. I have met several programmers that use major, commercially available image processing systems. When they add new processing capabilities to the system that their employer has purchased they all use the same basic technique -- start over!

This technique is fool-proof. You don't have to struggle through reams of misspelled documentation. You can avoid the originator's design flaws, and generate some new ones of your own. You are as free as Robinson Crusoe. You can keep your job as long as you like. And you can still blame your problems on the original supplier anyway.

5: Systems Design for Understandability - P1 thru A2

All data analysis systems, in particular the ones we describe here, require design, implementation and maintenance. If anything goes wrong during any of these three phases, the system will see a very short life cycle, and the end-user would probably have spent fewer dollars if he had used more traditional techniques.

Furthermore, it is better to say "I don't know" than to quote results that rest on more assumptions than any one person can possibly verify. What I am saying here is that one must have a very confident grasp of what these systems (especially A1 and A2) are actually doing to the data in order to specify a processing sequence and provide a lucid report of subsequent results.

Here, then, is the point of this paper and the goal of the system design that I am about to describe:

> The user interface is not only critically important, but it extends well beyond slick graphics, convenient cursor movement, and analyst's-console understandability,

although these are very important. Indeed, in the case of A2 systems (if not all levels), the programmer interface is exceedingly important. Of course in A2 systems the programmer is the major user. When he is done, his programs should run in A1 (turnkey) mode.

One cannot expect to simply hire a programmer to implement correct, fast, useable and understandable program modules that are to be integrated into an existing system that already has some number of similarily constructed modules without providing reasonably sophisticated development tools with which to work and document.

6: User Interface - a Definition

I hope you have noticed that I have avoided using the misleading phrase "user-friendly". These two words now cause any manager who has purchased an analysis system in the past three or four years to cringe. Although there is such a thing as user-friendly software, it hasn't yet made its debut in the image analysis world. No wonder. None of us really understood how we were going to get the analyst in touch with our programs until a body of well understood techniques and terminology could be established. Although that process is by no means complete, we are beginning to see the outlines of that body.

I have, on numerous occasions, heard comments from both managers and analysts on the weaknesses inherent in the user interfaces used in current work-station technology. My disussion, here, is meant to be a guideline for such managers and analysts so that they might be more demanding when specifying their future interface requirements.

ALL OF THE CAPABILITIES DESCRIBED HERE SHOULD BE CONSIDERED INDEPENDENTLY OF ACTUAL APPLICATION CODE. IN OTHER WORDS, THE USER INTERFACE SHOULD NOT BE EMBEDDED (TASKLINKED) WITHIN APPLICATION PROGRAM MODULES.

6.1: THE OPERATOR/ANALYST CONSOLE

The less hardware that the analyst has to manipulate, the better. Double keyboard systems are confusing. Spatial and image-display state control via data-tablet (or touch-screen) is preferred, but keyboard and data-tablet control must be integrated and the division of labour must be carefully defined.

All messages, prompts, menus and help documentation written to the terminal should be in concise, understandable

non-computerese. We should be using, as much as is possible, terminology from established literature rather than words invented by programmers.

6.2: ON-LINE HELP

All system functions should have sufficient on-line help to allow the analyst to come to the work station with only his data tapes and professional literature. This means that ALL user-level system documentation should be on-line. Getting it on-line is easy. Making it especially readable is a different story. The user/analyst should be able to enter further documentation, especially if he has written local procedural command sets for himself (or his colleagues). This documentation should be stored, accessed and displayed in a manner that appears identical to the documentation supplied with the system.

6.3: COMMAND PROTOCOL

Assuming manual keyboard control entry, all commands should be constructed from a well-formed syntax. You should be able to write, for example, a Backus-Naur form syntax description for the full set of commands. Extra points are awarded if the syntax fits on two pages; points are lost for syntax exceptions - which probably won't be used anyway.

6.4: PROGRAM CONTROL ACCESS

You should be able to set up ALL program control via COMMANDS and MENUS.

COMMANDS must be provided allowing the analyst to examine and change the state (value) of ANY control parameter for ANY program in the system without actually running the application programs. As much as is feasible, the interface should check user input for validity while s/he is setting them, rather than during execution of the application program.

All programs should have MENUS. Menus should be invoked, formatted and manipulated in a single, consistent manner. Dense menus should have two (not more) states: one that provides access to the basic functions for that capability, and another that provides advanced options. Nesting menus more than two levels deep is for the computer-literate, hence this should be avoided.

Reading between the lines, program control-parameter handling as I have described here is well served by data-dictionary methodologies. This approach provides, as fallout, immediate translatability of menus, messages and prompts to other natural languages.

All programs and analyst-written procedures must be dispatched for execution using a single syntax structure. In this way, supplied-programs and the procedures created for (or by) the analyst on-site will appear to be dispatched in the same manner - simply as commands to the machine to do something.

6.5: PROCEDURAL CONTROL

ALL program modules and program set-up commands must be dispatchable from automated command PROCEDURES. Such procedures are arbitrarily long sequences of system commands.

These procedures must be NESTABLE. This means that a procedure can itself dispatch another procedure. A caution is in order here. If you rely on your operating system utilities to program this capability you can't readily transport it to another system.

These procedures must have CONDITIONAL dispatch capabilities, one of the fundamental prerequisites of so-called 'expert' systems. This implies that in addition to an 'IF' statement, access to application control parameters and local arithmetic capabilities must be provided.

These procedures should have capabilities for internal COMMENTS, on-line HELP documentation and a way to show a control parameter MENU without actually dispatching the procedure for execution. (If these HELP documents and MENUS look like the ones provided by the actual application programs, so much the better.)

6.6: EDITING OPERATOR PROCEDURES

The interface should provide a mindlessly simple EDITOR to create such procedures. (The model for such an editor might look like a critical subset of the BASIC editor available on most personal computers.) The analyst should be able to STORE, RECALL and ALTER such procedures both temporarily and permanently.

All of this must be done using the straightforward syntax described earlier. The operator/analysts CANNOT be expected to slip in and out of the user-interface handler to the operating-system utilities and direct application program control.

THIS MEANS THAT EVERY KEY THAT THE OPERATOR/ANALYST TOUCHES WILL APPEAR TO BE INGESTED AND HANDLED BY ONE AND ONLY ONE PROGRAM. If you want to avoid serious maintenance problems, I recommend that there be, in fact, only one interface program.

6.7: HISTORY LOGGING

HISTORY LOGGING capability should include: turning it off and on, deleting it and printing it. The log should include time stamps for the purpose of execution timing, especially in A2 systems. If your system will provide the sort of procedural control that I have already described, it is not necessary that the log file actually be an executable entity in the manner of journal files.

6.8: ERROR HANDLING

ERROR HANDLING is probably the most difficult aspect of the user interface. The reason for this escapes many non-programmers, but it is due to the fact that there are several fundamentally distinct kinds of errors and we do not seem able to handle all of them in the same manner, although we should try to make it appear that way, if we can.

6.8.1 HARDWARE ERRORS. The user interface handler won't know about peripheral hardware other than disk and terminal, although the application package hosted by the interface might include exercise and diagnostic programs specific to displays, plotters, digitizers, and film recorders.

6.8.2 OPERATING SYSTEM ERRORS. The user interface handler should be designed to make the operating system transparent. Operating system errors should not show up at this level. If they occur during the execution of an application program, they should be treated as detectable programmer errors.

6.8.3 DETECTABLE PROGRAMMER ERRORS. In the case of detectable programmer errors, the application package error handler must produce a message (preferably hardcopy) that states enough, succinct information to enable the maintenance group (original supplier?) to solve the problem.

This typically includes the values of all of the control parameters so that non-data-dependent errors can be regenerated. Data dependent errors are harder - the image data generating the problem may have to be kept available.

6.8.4 NON-DETECTABLE PROGRAMMER ERRORS. In the case non-detectable programmer errors, the application program may behave very badly. Nasty program aborts, infinite loops, incredibly unreadable operating system error messages spew forth on the analyst's console. These problems are meant to be solved as they show up, usually by the embarrassed supplier or by frustrated in-house maintenance staff. Fortunately, mature software packages have few of this type of error.

6.8.5 DETECTABLE OPERATOR ERRORS. These consist of every conceivable thing the analyst might mistype at his console. The interface must catch as many of them as possible, without compromising the application independent nature of the interface handler. All error messages at this level might be kept in the control parameter dictionary - for translatability. These messages must be worded carefully and documented throughly.

6.8.6 NON-DETECTABLE OPERATOR ERRORS. This variety results from legal but non-sensical operator set-up. On-line documentation should be lucid enough for the operator/analyst to understand when he has made such an error, and how to recover from the problem - usually by restarting his procedure.

6.8.7 "LET'S STOP AND START OVER" ERRORS. These are the most important ones. If the analyst has kicked off an execution run and realizes that he hasn't set it up the way he meant to, then there should be some way, via the interface handler, to halt the entire process, recover files that are in unknown states, fix his error, and restart. All this must be done under the auspices of the interface, and without waiting for the error to run to its natural (or unnatural) completion. The solution to this problem requires a lot of thought, especially when running nested procedures. Usually a special application-dependent clean-up program or procedure must be invoked by the interface after actual abortion of the application process. Asychronous terminal I/O must be employed, since the keyboard cannot be left unguarded while an application process is executing under the interface. Some operating systems may not support the necessary capabilities.

6.9: PROGRAMMER INTERFACE

If the system designer has decided to opt for the sort of user interface that I have been describing, and if he decides to use a data dictionary approach such as is appropriate, then large parts of the programmer interface fall out as a by-product.

The programmer of individual modules NEVER needs to consider where, how and when he will prompt the analyst for control input. He doesn't have to compose messages and prompts. He doesn't have to loop back and reprompt if the analyst has typed an error. He doesn't have to clear the screen and write pretty menus.

In short, he can discard what typically turns out to be about 50% of his code, and simply say "GIVE-ME-PARAMETERS" to the control parameter dictionary. He may have to confirm that some of these parameters have reasonable values, because he will probably not want to confuse the interface with too much conditional value checking. (This keeps the interface application independent.)

All errors detected at this level will necessarily become fatal, but the interface is there to catch your analyst anyway.

7: The Expert Analysis System Interface (EASI)

PERCEPTRON has spent several man-years developing the user interface that I have just described.

EASI is:

- A HIGH-LEVEL USER LANGUAGE (BASIC LOOK-ALIKE)
- APPLICATION INDEPENDENT
- SPECIALIZED FOR INTERACTIVE SCIENTIFIC ANALYSIS
- A SINGLE SYNTAX USER INTERFACE
- A PROGRAMMABLE INTERFACE
- DESIGNED FOR DIRECT-TO-ANALYST USAGE
- MODIFIABLE WITHOUT REGARD TO THE APPLICATION PROGRAMS IT HOSTS

EASI provides:

- DEFINING AND CHECKING OF PARAMETERS
- EXAMINING AND EDITING OF PARAMETERS
- MULTI-LINGUAL MENUING AND PROMPTING
- A BASIC EDITOR SUBSET
- A PROCEDURE WORKSPACE WITH LOAD AND SAVE
- NESTING OF PROCEDURES (INCLUDING RECURSION)
- CONDITIONAL TESTING, BRANCHING
- LOCAL ARITHMETIC
- HISTORY LOGGING
- AUTOMATION OF ON-LINE HELP
- SPAWNING OF APPLICATION TASKS
- INTERACTIVE AND FOREGROUND MODES
- BACKGROUND AND BATCH MODES

EASI's optional, FORTRAN, programmer support library, IMP, provides:

- A SINGLE CALL TO GATHER ALL NECESSARY PARAMETER FROM THE DATA DICTIONARY
- CALLS TO READ AND WRITE PARAMETER VALUES FROM AND TO THE DATA DICTIONARY
- INTERFACE INTO EASI'S MENU/PROMPT CAPABILITIES
- AUTOMATIC ASSIGNMENT OF LUNS FOR ALL PERIPHERALS
- PROGRAMMER ERROR HANDLING ROUTINES
- SEMI-AUTOMATIC OVERLAYING (RSX-11M ONLY)

8: The Pattern Analysis and Correction Expert (PACE)

PERCEPTRON has spent several more man-years developing an image processing and analysis package that runs under EASI.

PACE is:

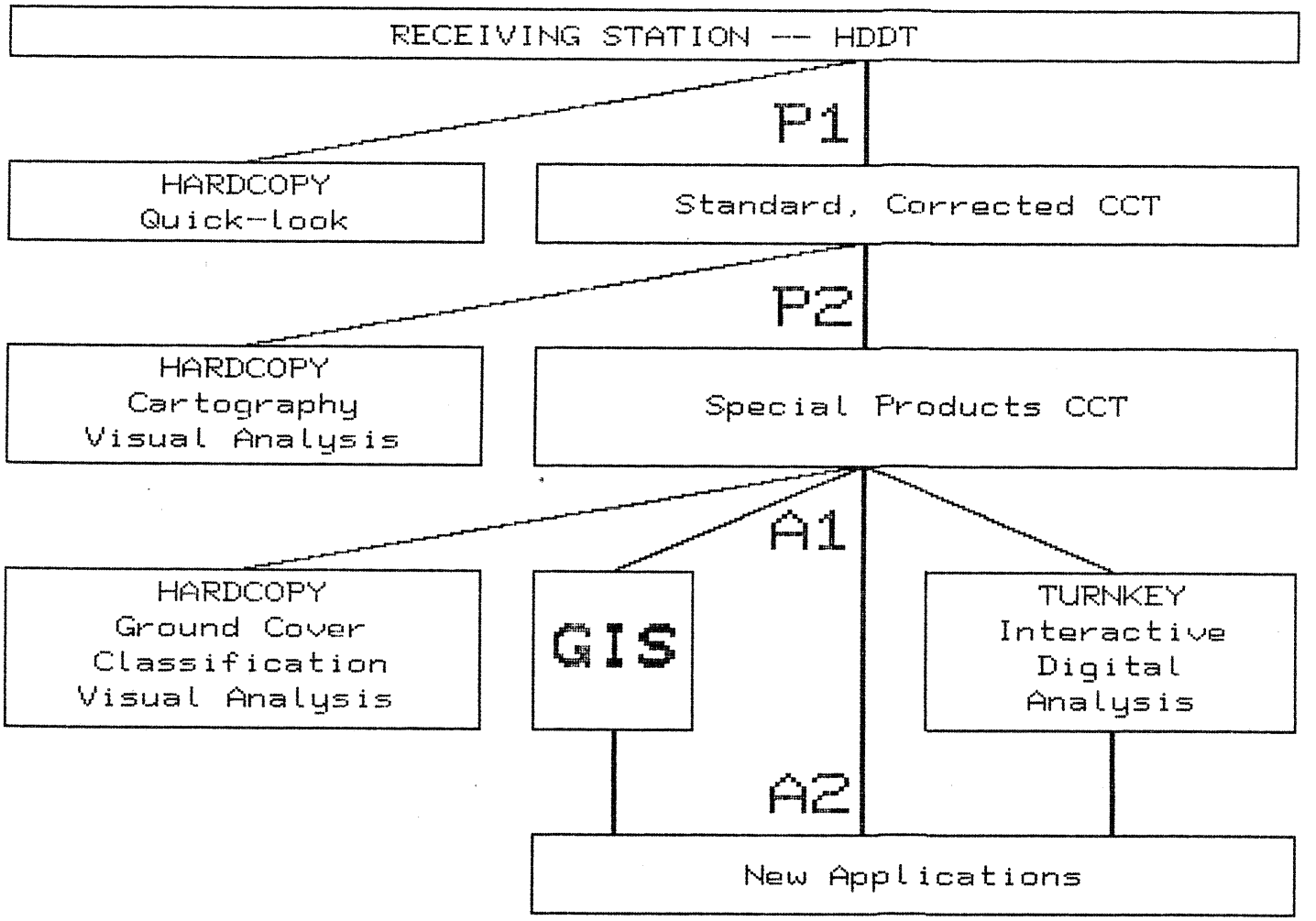
- SPECIALIZED FOR REMOTE SENSING ANALYSIS
- INTERACTIVE AND BATCH
- HOST AND DISPLAY INDEPENDENT
- DESIGNED FOR DIRECT-TO-ANALYST USAGE
- ANALYST-PROGRAMMABLE VIA THE EASI INTERFACE
- FORTRAN-PROGRAMMABLE WITHOUT MODIFICATION TO THE USER-INTERFACE

PACE provides:

- DATA INGESTION FROM ANY CCT
- FULL UNIDSK-84 ARCHIVING AND ENQUIRY
- DEVICE-TO-DEVICE IMAGE TRANSFER
- AN IMAGE FILTERING PACKAGE
- SEVERAL LOOK-UP TABLE ENHANCEMENT CAPABILITIES
- SEVERAL DATA TRANSFORMATION CAPABILITIES
- SEVERAL DATA CLASSIFICATION CAPABILITIES
- HISTOGRAMMING AND SCATTERPLOTTING
- SEVERAL IMAGE AND GRAPHIC EDITING CAPABILITIES
- IMAGE-TO-IMAGE GROUND TIE-DOWN FOR REGISTRATION
- IMAGE-TO-MAP GROUND TIE-DOWN FOR RECTIFICATION

UNIDSK-84 provides:

- UPWARD COMPATABILITY WITH CCRS' ORIGINAL UNIDSK-11
- A MULTI-CHANNEL DATABASE FILE FORMAT
- A SINGLE FILE STRUCTURE FOR IMAGE ANALYSIS
- A HOST-INDEPENDENT RANDOM (BLOCK) ACCESS FILES
- SINGLE-FILE STORAGE FOR ALL AUXILIARY DATA TYPES (TRAINING-SITES, HISTOGRAMS, MATRICES, LUTS, ETC)
- A CONCISE EASY-TO-PROGRAM SUPPORT LIBRARY
- CONCISE EASY-TO-TRAIN SOFTWARE APPLICATIONS
- HISTORY TRACKING, DATA LISTINGS, COMMON ACCESS SEARCHING, ARCHIVING, DELETION, UPDATING, REPORTING, ASSOCIATION, ENQUIRY, EDITING, ETC.



P1: First Level Production
P2: Second Level Production
A1: First Level Applications
A2: Second Level Applications

FIGURE 1: Four Classes of Digital Image Processing Systems.

The Operator-Analyst Interface

THE OPERATOR-ANALYST CONSOLE

ON-LINE HELP

COMMAND PROTOCOL

PROGRAM CONTROL ACCESS

PROCEDURAL CONTROL

EDITING OPERATOR PROCEDURES

HISTORY LOGGING

ERROR HANDLING

1: hardware
2: operating system
3: detectable programmer
4: non-detectable programmer
5: detectable operator
6: non-detectable operator
7: "LET'S STOP AND START OVER"

The Programmer Interface

Give me my control parameters

Check to see if OK

If OK - Do the job

Figure 2: Broad-Spectrum Definition of the USER INTERFACE

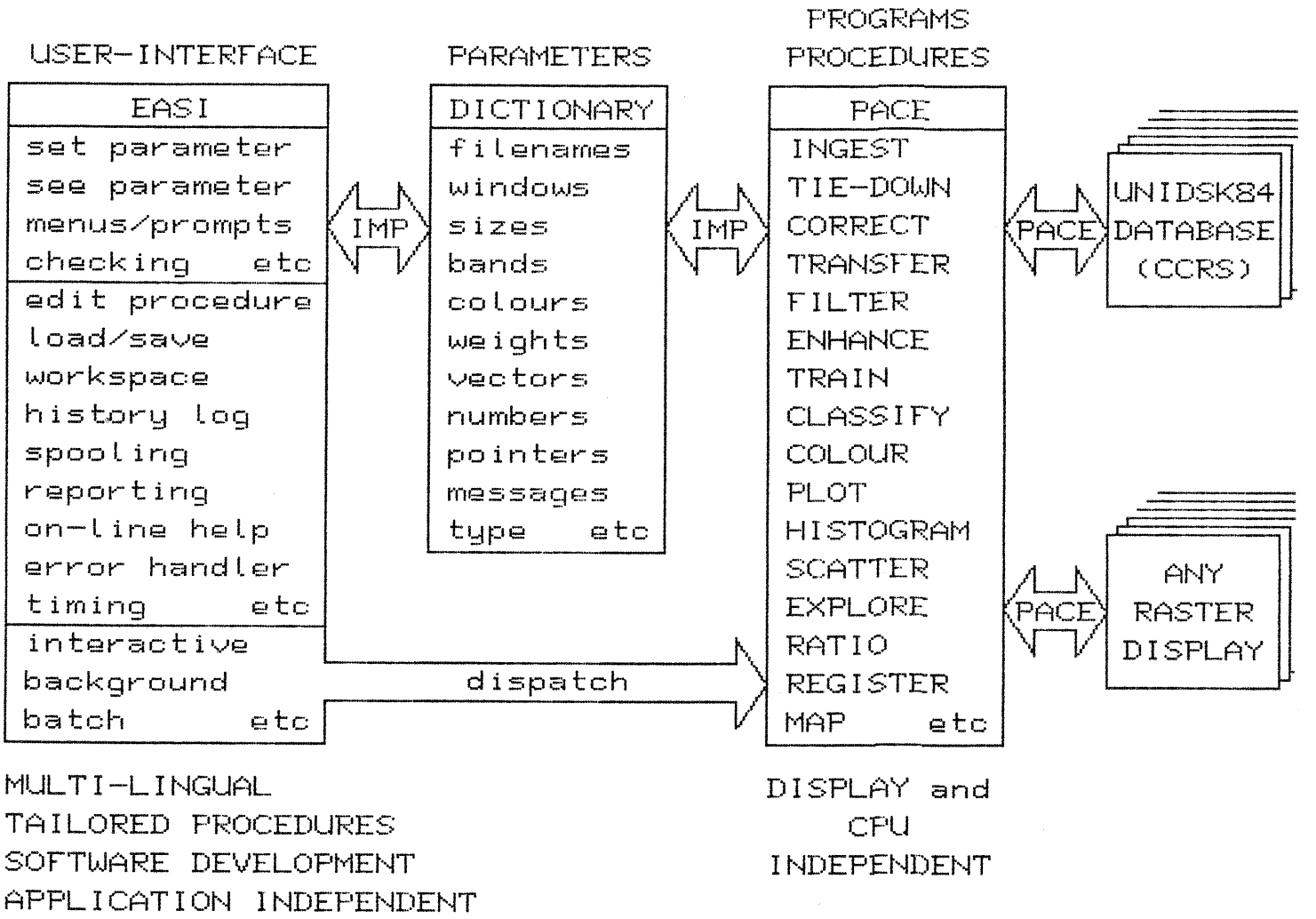


Figure 3: EASI/PACE - Structural overview