THE INTERACTIVE IMAGE SYSTEM FOR THE METEOSAT PROJECT

L. Fusco, G. W. Kerr, R. Powell, H. Rupp
European Space Agency
E S O C
Robert-Bosch-Strasse 5
D-6100 DARMSTADT (Germany)

1. Introduction

The Meteorological Information Extraction Centre (MIEC) of the ESA-METEOSAT Ground System at Darmstadt, West Germany, performs the extraction and dissemination of meteorological products generated from METEOSAT images.

The products (cloud displacements, sea surface temperatures, cloud top heights, radiation balance, cloud analysis, water vapour content) are generated in near real-time by automatically scheduled batch programs on a large mainframe computer. The products so calculated are then transmitted by inter-computer link to the MIEC Interactive Display System, where they are quality controlled by meteorologists before re-transmission to the mainframe for subsequent dissemination to the users.

2. Hardware Overview

The MIEC Interactive System is based upon two interconnectable mini-computers (NOVA 830), attached to three specialised meteorological display consoles and 30x64 Kbyte refreshing storages.

Each console consists of the following hardware units:
- A graphic display of 1024x1024 elements with absolute and incremental vectors, alphanumerics and 4 basic colours. This display is used for graphics and histo-

grams, and alphanumeric display of the results of MIEC processing. The display is equipped with a light pen for selection of options or picking of specific areas on the screen.
- An alphanumeric display used as a logsheet for indicating the current status of a process (image selected, coordinates of the displayed area, function in process...), and a scratch pad for all intermediate results the operator may want to keep.
- An alphanumeric keyboard for entering commands or text.
- Two function keyboards for initiating given functions
- An image display of 512 lines of 512 pixels. This display is used for image data display (with or without grids) and for software generated pseudo-images.

Each image display screen consists of a 512x512 dot matrix. Each dot is composed of 3 colours: red, blue and green. Intensity and colour are controlled independently by the values in corresponding 8-bit bytes of any two refresh memories (or by the same 8-bit byte from a single memory).

The processors for the colour and intensity channels are controlled by independent sets of operator controllable registers which allow masking, shifting, OR-ing, thresholding and scaling of the byte values from each memory.

The colour tables are 256-byte, software-modifiable, look-up tables, indexed by the digital output from the colour-channel processor.

3. Software Overview

The interactive SW is run on a NOVA 830 under the Data General multi-tasking, dual programming operating system RDOS REV.3. This operating system permits two programs to run in parallel - the foreground program, and the background program. A program may consist of a number of tasks, either resident or overlayed.

Overall system control is handled by a set of permanently resident tasks (the 'root' tasks), while each console is individually controlled by tasks loaded into fixed overlays, one overlay per console. A special task in the root (the console root task, or CRT) monitors the alphanumeric keyboard of each console.

The operator may type in commands which result in an overlay being loaded in the overlay area assigned to his console, and one task inside this overlay being executed. The procedure command language described in section 4.1 (the MIEC command language, MCL) controls this process.

The program performing this multi-task, overlayed console control is the so-called background program. In addition to the normal SW facilities available via the background program, any console operator may obtain additional SW facilities by initiating the foreground program via a task in his overlay in the background.

The foreground program is controlled by a function keyboard. The operator can continue to run tasks in his background overlay asynchronous with operation of the foreground program.

4. Design Features

The design features of most interest can be classified under 2 main headings:

- the command language,
- the picture concept.

4.1 The Command Language

Interactive processing is controlled by invoking sequences of independent, user-overlay tasks, initiated by means of language commands. The commands may be input either directly and individually via an alphanumeric console by the interactive operator, or (as for normal operational running) from indirect command files containing pre-edited chains of commands (called procedures), or both.

Processing paths through the commands in a procedure are controlled at execution time by:

(a) operator decision, via both function keyboard and/or alphanumeric input
(b) results of previous processing actions.

The command language allows many of the elements of a high-level programming language, including:

- forward and backward jumps
- variable count loops (with loop nesting)
- arithmetical and logical testing (and conditional jumps)

- procedural nesting (called sub-procedures)
- parameter passing via software registers
- indirect parameter files
- register arithmetic.

Each overlay task performs operations according to parameters and data sets supplied at run time, either directly from the operator, or from indirect files, or from SW registers (or any combination).

Tasks which require run time parameters call a resident subroutine in the root, supplying to this subroutine a list of parameter names and types, upper and lower limits of each parameter (integer parameters only), and default values, if applicable. This routine, which is part of the command language interpreter, reads parameter values from the current input device (alphanumeric keyboard, or indirect file), updating any input parameters corresponding in name to a parameter in the supplied list, until a double slash (//) is detected, after which control is returned to the user task. (The order of parameter input is unimportant. Parameters in the task list, which are not input at execution time, retain their default values.)

4.1.1 Outline Structure of the Language

The logical unit of input to the command language interpreter is an atom having one of the forms:

1. Command word
2. /± logical variable
3. /parameter = value
4. /parameter = \software register n

Character strings in the command line buffer are semantically interpreted according to one of these forms until a 'carriage return' is detected, after which the system determines the physical source of input for the next physical line (indirect file, or alphanumeric keyboard), and refills the command line buffer.

1. Command words may be of the form:
   - task name
   - [indirect file name
   - /*

   Logical input to a given task is terminated when the command language interpreter detects either // or a task name, as the next character string in the command line buffer. (The logic of a task may, of course, contain any number of requests for parameter

input, each request being terminated when // is encountered.)
The command word - [indirect file name - requests the system to obtain the next and subsequent lines from the corresponding ASCII file until either EOF, or /* is encountered. If EOF is encountered the next physical line is obtained from the previous source of lines (which could itself be either an indirect file, or the alphanumeric keyboard).
The command word - /* - requests the system to obtain the next logical line (to //) from the alphanumeric keyboard (valid only inside indirect files), after which return is made to the indirect command file.

2. The atom - /± logical variable - attributes the value true or false to the variable, according to the sign + or - respectively.
3. The atom - /parameter = value - sets the parameter equal to 'value', where 'value' may be numeric or string.
4. The atom - /parameter =\software register n - sets the parameter equal to 'content of SW register n'. (Software registers may be set and/or tested both inside tasks and in procedures. The registers are stored in the root area and are therefore global to all tasks. Values may be either integer or character, depending on prior definition of the register type.)

4.2 The Picture Concept

To construct a screen picture, the operator can define which memory should control picture intensity and which memory should control colour. Additionally, the operator can modify hardware parameters of the TV screen, and of both memories independently (e.g. masks, thresholds, scale factors, zoom, offset, etc.). Thus from one pair of memories, he can construct a wide variety of images on the TV screen.

The software has been designed in such a way as to remove from the operator the need to remember all the various parameter settings and refresh memory contents.

The basic concepts underlying the software design are as follows:

- any set of 256x256 - or 512x512-byte image data can be associated with an arbitrary mnemonic name, called a 'data-set' name

- any screen image can be associated with an arbitrary mnemonic name called a 'picture' name.

Pictures are in general composed either of one or two data-sets, each of which is loaded into a separate memory. All the hardware parameters associated with each picture are held on a file called the 'picture handler' file. Thus, once the operator has loaded his data-sets into memory, constructed his pictures, and saved the screen and memory parameters on the picture handler file, he can at any later time recall particular screen images simply by specifying the corresponding picture name to the task responsible for setting up the hardware registers of the display.

The physical memories used to hold the data-sets are allocated transparently to the operator, so that he need never concern himself with actual memory numbers.

The operator (or procedure writer) has a number of facilities available to simplify the task of handling images. These facilities fall into 5 main classes, as follows:

- tasks which create picture handler file (PHF) entries, allocate refresh memory and NOVA disk file space for the corresponding data-sets if required, and which generate new PHF entries from existing PHF entries
- tasks which obtain image data-sets from the mainframe and store them in pre-allocated NOVA disk image file space
- tasks which copy data-sets from NOVA disk file to refresh memory
- tasks which send data-sets from the pre-allocated image file space to the mainframe
- tasks which set up pictures on TV, according to picture definitions in the PHF.

5. Procedures

A procedure is in essence an ASCII file containing overlay task calls and parameter lists, interspersed with procedure-control commands.
Procedures can contain any number of task calls. Operational procedures typically contain 500-1000 command lines, each of which will normally initiate a 4K overlay task.

One of the major assets of the language is the ease with which large error free procedures can be developed.

The language elements invoke independent tasks which perform specific jobs of work and then terminate, and which are essentially independent of preceding task calls. Extensive trace facilities allow processing paths through the procedures to be instantly followed. Incorrect procedure logic can be corrected in minutes by on-line editing.

Each task is developed and tested as an independent entity, so that once it performs correctly, it can be used in a procedure without any possibility of interference with other tasks.

Typically, to write, edit and test a procedure of the order of 100 command lines, takes of the order of 1 to 2 days.

More than 60 independent task calls are currently at the disposal of the procedure writer, so that he can effectively develop and test programs of up to 200 Kbytes in a couple of days.

6. Application Tasks

In addition to the procedure control and picture handling tasks already mentioned, the following classes of tasks are also available:
(1) Hardware manipulation
(2) Colour table generation
(3) Animation control
(4) Array manipulation
(5) General facilities.

6.1 Hardware Manipulation

These tasks allow operator control of every hardware feature, via either keyboard, alphanumeric, potentiometer, or tracker ball input.

6.2 Colour Table Generation

These tasks supply easy to use facilities for constructing colour tables on-line according to the data types (e.g. image data or bit-planes), and dynamic ranges involved.

6.3 Animation Control

Animation consists in displaying picture sequences on a TV screen in rapid succession. Animation loops are used mainly to study short-term phenomena and for interactive quality control of the various meteorological products- for example, wind vectors, sea surface temperatures, etc.
The system has been designed to allow easy setting up and control of animation loops.

The following features are available during an animation loop:
- zoom any or all of the pictures (keyboard control)
- offset any or all of the pictures (tracker ball control)
- change speed of the animation (potentiometer control)
- change thresholds and scale factors of intensity and/or colour channels (potentiometer control)
- change direction of the loop (keyboard control)
- step backwards or forwards through the loop (keyboard control)
- change display time of each individual picture in the animation (keyboard control).

Note that offsetting can be performed on a zoomed sequence, so that the effect is similar to that of sweeping over an area with a camera fitted with a telescope lens.

6.4 Image Manipulation

A number of facilities exist for performing array manipulation on the refresh memories, including convolution on any defined area of one single data set picture, between the image data and a defined window.

6.5 General Software

These tasks may be grouped as follows:
- file handling tasks, to print, delete, change attributes, edit, list, rename, etc. standard NOVA files
- data handling tasks, to convert data from the M/F standard format to NOVA data format, and to proprocess card image files
- procedure preprocessor task, to preprocess procedures written in macro form.