APLITDS: An APL Development System

Carlo Alberto Spinicci
APL Italiana srl
via Eustachi, 11
20129 Milano
Italy

Abstract

In spite of its tremendous expressive power, APL lacks a convenient environment for developing large, interactive systems. The workspace structure prevents easy sharing of code and the realization of user interfaces is at least as difficult as in other languages. This paper describes an integrated development environment that attempts to overcome these limitations.

Introduction

APLITDS is the development system used by APL Italiana for internal development. The awful name is an acronym for the APL ITaliana Development System. We never found a better name and eventually grew accustomed

Development of APLITDS started in mid-1987 as we tried to organize the large number of utilities for file handling, screen definition, reporting and graphics which we had accumulated during the three previous years of APL*PLUS/PC usage. The first version of APLITDS was released after a couple of months of programming in spare time. From start to finish, the implementation of APLITDS has been a one-man job - unfortunately, my own.

The main aims of APLITDS are the same as any system of this kind: to reduce development efforts, to simplify maintenance, and to provide for uniform programming styles and user interfaces.

The Workspace Problem

The standard APL environment, centered around the workspace, is perfect for casual users, ad hoc solutions and small personal applications, but it falls short for developers of complex, professional, systems. It is simply too difficult to ensure that all the application workspaces use the same (latest) version of the utilities. Further, it is not practical to have a copy of all your development tools in every workspace and)COPYing and)ERASEing everything each time is slow and annoying. This problem has been highlighted many times in the past [Ber84] [All86] and numerous solutions have been proposed.

APLITDS keeps programs in files (so that different applications can make direct use of them) and uses only one (small) saved WS to boot the system.

Building User Interfaces

Probably most of the code written for large, interactive systems is devoted to managing the user interface. In this respect APL doesn't provide much more than any other language. Screen tools included with most APL interpreters are at a rather basic level, as AP124, AP126, □WIN, and so forth. Even the more advanced tools such as Dyadic's □SM [Cur89] handle only the final part of the task. A lot of code has to be written which has very little to do with the application itself, and concerns itself with the (graphic) input and output technology used.

One of the beauties of APL immediate execution mode is that the user is at the very top of the hierarchy with all the programs at his fingers. When using traditional interactive programs written in APL or any other language, the user is just a subroutine of the system and she/he may only react in ways foreseen by the programmer. This "inverted" hierarchy tends to produce user interfaces that frustrate the user and promote clumsy programming. The solution is perfectly highlighted in [Kro90]: "Instead of letting the application call the tools, let the tools call the application". This is what we tried to do in APLITDS.

Structure of APLITDS

The main components of APLITDS are:

- a data dictionary, where all data variables used in the application are described.

- a screen editor for both fill-the-blanks and spread-sheet style forms (forms are filed on disk and retrieved at the moment of execution). The definition of the variable part of a form is dictionary driven.
- an overlay system for storing functions and variables, and retrieving them when needed.
- a report generator.
- a business graphics generator.
- a set of utilities.

An APLITDS "screen" is not only the basic unit of interaction between the user and the application, but also the fundamental element of APLITDS programming. This "screen oriented" programming is typical of certain application generators such as PRO-IV [McD88].

Interactive applications usually follow this general pattern:

- Data is read from files.
- Computations are performed to obtain all required information.
- Data is formatted, merged with fixed text, and presented to the user.
- Some code for managing the interaction with the user (as □WIN) is executed.
- User input is validated.
- After some final computation, data is written to file.

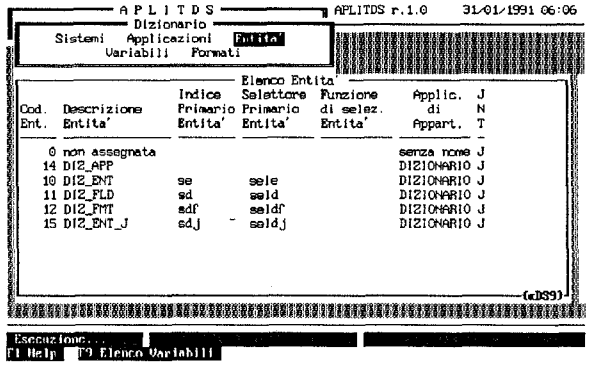

fig 1. List of tables

In APLITDS the above activities are defined in the data dictionary. The activities are performed at the appropriate moments when the user make use of the screen. Some of the definitions are APL expressions, and thus the data dictionary defines how APL functions are called from the screen.

APLITDS screens give the user a great deal of freedom in managing data: she/he can select and sort rows, aggregate numeric data, define and print reports, and create graphs. She/he can also control some aspects of the screen layout, such as selecting and ordering columns of information to appear on the screen from a predefined set. This works - at a keystroke - on a thirty rows of a memory-resident table, or on several thousand rows of data in a file.

Furthermore, all this is obtained without any application code, it's just a "fringe benefit" coming from the screen definition and the data dictionary.

APLITDS Data Dictionary

Variables are the basic objects described in the dictionary. They can be APL workspace variables, or the results of computation. They can reside in memory, or on a file (not necessary an APL file, you can use any file system which can be accessed from APL).

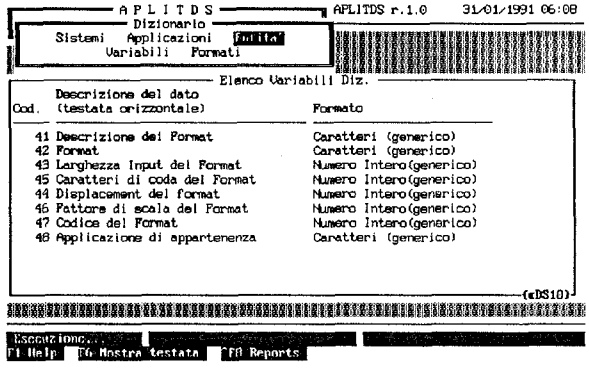

fig. 2 List of fields

For each variable, the dictionary contains the following information:

- A short description of the variable.
- The format (□FMT syntax).
- The header to use in spread-sheet like forms and in reports (it can be fixed, or calculated at execution time).
- The APL expression that calculates (or retrieves) the value of the variable.
- The APL expression that produces the external representation of the variable from its internal value (e.g., from internal date format to 'yy-mm-dd').
- The APL statement used to accept input for the variable. APLITDS itself supports several data types such as character strings, numbers, dates or items selected from a predefined set of values. If you have special requirements, the expression can do anything you need, even opening a completely new screen.
- The APL expression that validates input.
- The APL statement that stores the input (e.g., assigning it to a variable or writing it to disk).
- How to sum values of this field. Certain fields (especially calculated fields such as averages) cannot be totalled using a simple plus reduction.
- Various less important attributes.

Variables are organized in tables (very similar to relational tables). For tables, the dictionary describes various operations on the table.

The most important information recorded for a table is:

- The APL statement that, when executed, adds a new item (record) to the table.
- The name of a variable to be used as a "selector" for the table (it will contain the indication of which records of that table are to be considered "selected").
- The name of a variable to be used as a pointer for the table. The system will set the value to the indices of those records which are involved in the current operation. Most of the expressions which describe APLITDS variables will be sensitive to this value.
- The {optional} name of a function, which does the setup needed before referencing the table (such as opening files).
- The {optional} name of a function, which does the setup needed before any operation on the currently active items (such as reading from files).
- A set of expressions that join this table with others. These expressions compute the pointer of one table from the pointer of another. They will be used by the system when the same screen or report contains variables belonging to different tables.

The dictionary also contains descriptions of standard formats used through the application, and complementary information regarding applications and their related data and program directories. Dictionary maintenance is performed using interactive APLITDS screens.

The Screen Editor

The screen editor of APLITDS enables the programmer to define three types of objects:

- Fill-the-blanks screens. A fixed format screen with a constant part and variable fields. The constant part of the screen is designed by the programmer using an interactive utility. The variables, described in the dictionary, are then positioned on the screen using the same utility.
- Spread-sheet style screens. A collection of variables (columns) for all the selected records (rows) of the main table of the screen. Vertical and horizontal scrolling is allowed. Constant parts (headers) are taken from the data dictionary as well as any information needed for their handling. The developer is only requested to specify which variables are to be included.
- Menus. A list of choices and related APL statements to be executed. The menu is automatically formatted. The user can activate one choice moving the cursor and pressing the "enter" key or just pressing the first uppercase character of the choice.

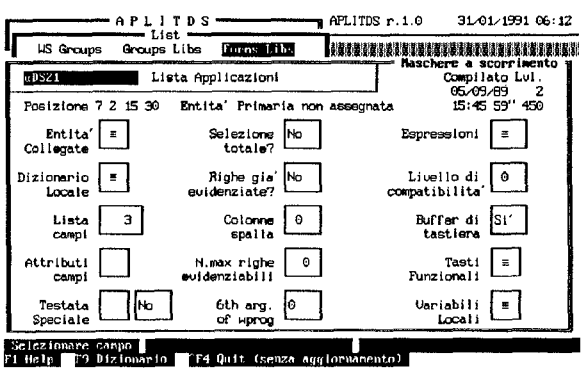

fig. 3 Screens editor

Both types of screens are filed in screen libraries. Menus are kept as packages together with APL functions and variables in the function libraries (see below). Editing of screens is also performed using interactive APLITDS editors.

Screens can have local variables and a local dictionary (we have developed systems using APLITDS, which make no use of the central application dictionary).

A set of APL statements can be defined so that they will be executed at useful points such as during the initialization of a screen, before or after the input of a field, or before the display of a new item. Using these statements together with dictionary statements it is possible to design applications where the flow of operations is completely controlled by the screen definitions.

Function keys can be defined for screens and menus. The APL statement to be executed and an optional text to be displayed on the last line of the screen are recorded for each function key. "Hot keys" (always active during an application) are defined as specially named functions. In this case, the informative text used to automatically create the help screen - always avalaible through the PF1 key - is kept in a public comment in the first line of the function.

At any moment privileged users can press a key to open an "APL immediate execution" window and perform any APL operation. The depression of another key resumes normal APLITDS operations. Screens, PF-keys and error trap definitions are preserved during the APL activity. Another useful key (not active in the final runtime applications) enables editing and recompiling the currently active screen. The application dictionary can be also updated "on the fly".

Function Libraries.

Programs and constants are collected in "groups" (packages). Groups are organized in "libraries" (APL files). Library names are single characters. Group names are three characters long. To refer to a group within a library, you simply catenate the library name to the group name (e.g., DWMB is the name of the group WMB inside the library D)

Libraries with uppercase names are general libraries. Libraries with lowercase, or numeric names are "application libraries". The actual file name for libraries is built

from the letter that designates the library plus a fixed part for general libraries and an application dependent part for application libraries. Screen libraries have the same naming convention.

A single function "uG" does all the housekeeping of groups and libraries. The statement uG'G DWMB' defines the group D WMB in the workspace, while uG'S DWMB' stores a modified version of the same group into the D library. Other operations recognized by the uG function let you list the contents of libraries and groups, create new libraries and groups, etc.

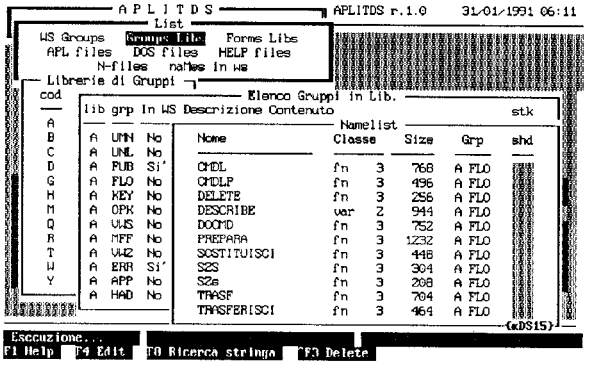

fig.4 List of the variables of a group

When a group is stored, the uG function adds, or updates a standard comment in each function of that group that specifies when the function was last changed, and who changed it (changed functions are recognized through a CRC code recorded in the same comment line). Management of groups and libraries can be achieved directly through the uG function, or through APLITDS screens.

Only the uG function, the package functions, and some error handling tools are kept in the boot WS. All other functions and variables are stored in libraries.

The Report Generator

At any moment, when using a spreadsheet screen, the user can create and print reports. Reports are defined by just selecting which columns are to be included. Partial totalling is permitted. A number of parameters let you control several aspects of the report such as suppression of detail (i.e., print only totals) for all rows, or just for a selected number of rows, printing attributes for totals, print destination (printer port or file), etc.

Escape sequences to drive the printer are fetched from a configuration file and can be modified.

Graphics

At any moment, when using a spreadsheet screen, the user can create and print charts. To obtain a graphic image of the data, the user has only to specify the variable (or variables) to plot, and the ordinate (one or two variables, not necessarly numeric), and then to select the kind of chart she/he wants (line, points, bars, pies, percentage bars, skyscrapers, line3D, or points3D).

Graphics can be printing on impact and laser printers in different orientations and dimensions. Charts are plotted using the □G* functions supplied with APL*PLUS/PC.

Utilities

A wide set of utilities, ranging from development tools to DOS interfaces, have been added to APLITDS.

The following list summarizes some of the activities that are possible through those utilities:

- Searching APLITDS libraries.
- Editing help files.
- Capturing a TIFF image from the screen (both graphic and text screens) for documentation purposes (all figures - with the exception of fig.5 - in this document were created using this tool).
- Creating a TIFF image that graphically shows which sequence of menu selections and PF keys the user has to follow to activate a certain function of an application (we use this option when writing manuals, see fig.5).
- Creating "auto demo" versions of an application to be used during exhibitions.
- Maintenance of our propietary data bases.
- Issuing a DOS command on a selected list of files.
- File (de)compression and archiving using PKZIP from PKWARE.
- Housekeeping of APL files.
- Controlling various printers.
- Changing screen attributes and keyboard layout.
- Creating (compiling) the runtime version of an application (i.e., compressing functions using STSC DIAM and UNCOMMENT utilities, extracting from libraries only those groups that are referenced in the application, and stripping any development specifications from screen definitions).
- Porting of a new version of an APLITDS application from the APL*PLUS/PC environment to APL*PLUS II and, preserving those functions, that have been modified in the new environment.
- Maintenance of different versions of the same application and multilingual versions of any message embedded in the application (this is still under development).

Some of these activities could be done using DOS utilities such as Norton Utilities or PC TOOLS, but using these "external" systems requires leaving the development en-

vironment, while the APLITDS utilities are immediately available to the APL programmer.

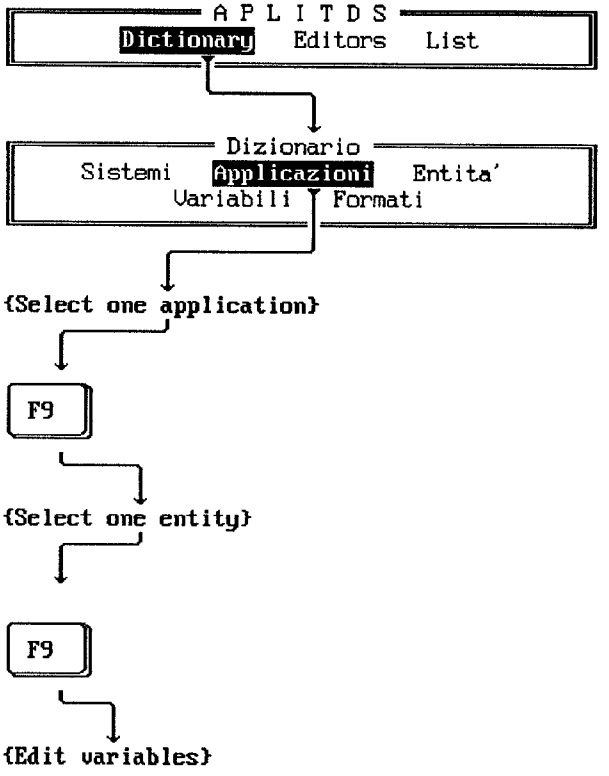

fig.5 How to edit a variable - Example of documentation

...more

There are other features of APLITDS, that have proven to be useful. The most frequently used is the ability to schedule background activities. A scheduled activity is an APL function to be executed at certain points, for example while waiting for user input, or during time-consuming operations. The most common of these activities (automatically scheduled by APLITDS) is the refreshing - every 60 seconds - of the time stamp showed at the upper-right corner of the APLITDS screen.

One of the applications written by APL Italiana using APLITDS is a PC front-end for mainframe applications based on a Sharp APL system and accessed through a telephone line. This PC application makes heavy use of scheduled activities, for example to let the user browse part of the message already received, while new material is constantly appended to the end of it. Unsolicited input (as in BBS systems) has been emulated on a half duplex network (such as the IPSA/Reuters network). A scheduled activity occasionally polls the remote system for new information while the user is free to browse everything that has already arrived on his PC.

Careful writing of the scheduled functions can reduce the interruptions experienced by the user to a negligible level.

Notes about Implementation of APLITDS

APLITDS is completely based upon itself. Any screen used in APLITDS (editors and utilities) is written using APLITDS and its dictionary.

Any facility that APLITDS provides to the end user, can be used by the developer during his work (the reverse is also true unless the facility is excluded during compilation of the runtime version).

APLITDS was written in APL*PLUS/PC using most of the features of this interpreter. This means that porting it to other APL systems would require a substantial effort. We have ported a complete version of APLITDS to APL*PLUS II and a very reduced subset (only the library management subsystem) to Sharp APL.

Applications Implemented using APLITDS.

We have implemented a large number of applications using APLITDS. Some of them are used internally to record our activities, or to maintain our customer data base, etc. Some applications were developed under contract, such as the PC front-end mentioned above, that has about one thousand installations. Others are "products" such as a portfolio management system for stocks and bonds called SOFIA, which is the the primary product sold by APL Italiana. Some of those applications are installed in Local Area Networks and allow multiple users to access and update the same data bases simultanously.

Conclusion

The development of a system such as APLITDS, using a powerful language such as APL is not impossible, even for a small company like APL Italiana. Creating a well documented, bug-free product from it is probably beyond our capabilities (or rather our capacity).

A flexible and extensive application environment capable of imposing a uniform style on all the products of our company, without stifling creativity, is of enormous value to us. Tasks such as maintenance and documentation are greatly simplified (our user manuals are at least 50% iden-

tical). This is true despite the fact that we are an APL-based company!

Screens provide an intermediate hierarchical level over computational (APL) functions. This brings APL programming closer to the main stream of non-APL programming facilitating the use of emerging analysys techniques and related CASE products. Screens and fields are objects which can be activated by the user. The environment knows how to react to a user actions. A simple reorganization of the application definitions could provide a framework for an object-oriented programming style in APL.

APLITDS itself is probably is not an example of good APL programming style. We have added new features each time we needed them, with the functionality which was required at that moment. I would expect that the development of similar (hopefully better) products, maintained by the major APL vendors would help to spread APL, giving a uniform base to the applications written in APL by different companies. This would lead to a more homogenous culture among APL programmers, which in turn will help the circulation of people and ideas within the APL community, which unfortunately remains very small.

References

[All86] D.B.Allen, L.H. Goldsmith, M.R.Dempsey and Kevin L.Harrel, *LOGOS: An APL Programming Environment*, APL86 Conference Proceedings, p.314 Manchester (1986)

[Bar90] Guy Barker, Douglas j. Keenan and Herman van Loon, *Conscientious Programming Using PMA*, APL90 Conference Proceedings, p.18 Copenhagen (1990)

[Ber84] Michael J.A. Berry, *Shared Functions and Variables as an aid to Application Design*, APL84 Conference Proceedings, p.57 Helsinki (1984)

[Cur89] A.D. Curtin and J.M.Scholes, □SM: A Full-Screen Manager for Dyalog APL, APL89 Conference Proceedings, p.107 New York (1989)

[Kro90] Morten Kromberg and Martin Gfeller, *An Application Develepment Platform*, APL90 Conference Proceedings, p.217 Copenhagen (1990)

[McD88] McDonnell Douglas, *PRO-IV Reference Manual*, Version 1.5, St. Louis, (1988)

Example of use

This simple example implements an invoice data entry system. Data is kept in the workspace. Customer names and addresses are kept in two character matrices: CUS and CUSA. Invoice headers and details are in two numeric matrices: HDR and DET. The first column of HDR is a pointer to customers, the first column of DET is a pointer to HDR. PROD is a character matrix containing product names.

The first step is to define the needed dictionary as follows:

```
TABLES:

Table: CUSTOMER    (Customers)
  Insert:   CUS←CUS' ' ◇ CUSA←CUSA' ' ◇
            ◇ selc←selc,1↑⍴CUS
  Pointer:  sc
  Selector: selc

Table: INVHDR      (Invoice Headers)
  Insert:   HDR←HDR0 ◇ selh←selh,1↑⍴HDR
  Pointer:  sh
  Selector: selh

Table: INVDET      (Invoice Details)
  Insert:   DET←DET(¯1↑⍴DET)↑sh ◇
            ◇ seld←seld,1↑⍴DET
  Pointer:  sd
  Selector: seld


FIELDS:

Fields in the table: CUSTOMER

  Customer name
    Output:   CUS[sc;]
    Store:    CUS[sc;]←wx
    Input:    wx←wIc

  Customer address
    Output:   CUSA[sc;]
    Store:    CUSA[sc;]←wx
    Input:    wx←wIc


Fields in the table: INVHDR

  Customer
    Represent:CUS i0
    Output:   HDR[sh;1]
    Store:    HDR[sh;1]←wx
    Input:    wx←wIs'CUS'

  Date of invoice
    Represent:dCVO
    Output:   HDR[sh;2]
    Store:    HDR[sh;2]←wx
    Input:    wx←wId 11


  Terms of payment
    Represent:dCVO
```

```
    Output:   HDR[sh;3]
    Store:    HDR[sh;3]←wx
    Input:    wx←wId 11


Fields in the table: INVDET

  Product
    Represent:PROD i0
    Output:   DET[sd;2]
    Store:    DET[sd;2]←wx
    Input:    wx←wIs'PROD'

  Quantity
    Output:   DET[sd;3]
    Store:    DET[sd;3]←wx
    Input:    wx←wIn 0

  Price
    Output:   DET[sd;4]
    Store:    DET[sd;4]←wx
    Input:    wx←wIn 0

  Row Amount
    Output:   ×/DET[sd;3 4]
```

Notes:

- Represent and Input statement, if omitted, are empty.
- `X i0 Y` is equivalent to `X[Y]` (in zero origin)
- `dCVO` transforms the internal representation of dates
- to the external numeric representation DDMMYY
- `wIc` inputs character strings
- `wId` inputs dates (11 is the external format used)
- `wIs` allows selection of one value from a list
- `wIn` inputs numbers

After defining the dictionary we will proceed with screen definitions. We will define three screens:

```
Screen α0S1: Customer List
Primary Table: CUSTOMER
Fields:
  Customer name
  Customer address

Screen α0S2: Invoice List
Primary Table: INVHDR
Joined Tables: CUSTOMER
Fields:
  Customer
  Date of invoice
  Terms of payment
  Customer address
PF keys:
  Key: 2  F2 Details
  Exp: seld←WHERE DET[;1]=sh ◇
       ◇ 0 0ρw'α0S3'

Screen α0S3: Invoice Details
Primary Table: INVDET
Fields:
  Product
  Quantity
  Price
  Row Amount
```

Notes: Function "w" opens the screen named in the right argument.

We will define one menu:

```
Menu sMEXAMPLE:  APL91 example
Choices:
  Customers      0 0ρw'α0S1'
  Invoices       0 0ρw'α0S2'
```

And finally an APL function to start the system:

```
    ∇ DO
[1]   APL91 example start function
[2]
[3]   banner          banner
[4]   uG 'gDWMB'      load the APLITDS
[5]   uG 'gAFAA'      screen manager
[6]   uG 'gAFSS'      and other basic
[7]   uG 'gAFUB'      groups
[8]   wPF             initialize screen
                       management
[9]   wSKI            Initialize skeduled
                       activities
[10]  'wTS' wSKD 1 0  Start clock as a
                       skeduled activity
[11]  READDATA        Reads CUS CUSA HDR
                       e PROD
[12]  wM 'sMEXAMPLE'  Start the menu
[13]  STOREDATA       Stores CUS CUSA HDR
                       e PROD
[14]
[15] ° (c)APLIT 0 INI spi 19/04/1991 10:37
                                      ¯2805
    ∇
```