Terminal transparent display language (TTDL)

by CARL E. KREBS, C. BUMGARDNER and T. NORTHWOOD

INCO, Inc.
McLean, Virginia

ABSTRACT

Terminal Transparent Display Language (TTDL) is a software language that implements complete and effective communications in an on-line computer system containing two or more different types of terminals. TTDL integrates differing terminal display techniques into a common language, freeing the application programmer from the specifics of terminal data handling and allowing him to concentrate on his primary function of providing a service to the terminal user.

INTRODUCTION

INCO, Inc. developed and is currently implementing Terminal Transparent Display Language (TTDL) as part of the Terminal Oriented Support System (TOSS) Research and Development project for the Rome Air Development Center (RADC). The purpose of TTDL is to provide a software language and support system which will allow communications to different terminal types without reprogramming. See Figure 1. Under TTDL, terminal differences become "transparent" and do not require consideration by the programmer.

TTDL was specifically designed for use on Digital Equipment Corporation (DEC) PDP-11 series minicomputers that are terminal-oriented. TTDL provides the capability to operate any number of different terminal types on this system because it approaches terminal transparency from a functional standpoint. TTDL was designed to support the logical functions required by the intelligence analyst, rather than the specific terminal devices he might wish to use. For example, TTDL allows the programmer to specify an "accentuation level" to emphasize a particular section of text. This level is then mapped into a specific terminal's physical characteristic; this may be blinking on one terminal, but may be reverse video on another terminal.

This paper addresses four specific areas of the TTDL system. Firstly, it describes the Display Specification Language, the primary communication tool of the application programmer.

Secondly, it describes the seven most important functional features of TTDL. They are:

- Screen Area
- Command Line
- Select Menu
- Screen Formatting
- Data Checking
- Control Input
- Display Library

Thirdly, it describes the separate software modules which drive the system. And lastly, it details the control and flow of data through the system.

By describing TTDL in terms of its user functions, this paper hopes to present a more meaningful software description; one from which the reader can understand TTDL's operation as well as its theory.

DISPLAY SPECIFICATION LANGUAGE (DSL)

TTDL defines displays using the Display Specification Language (DSL). This language is designed for use with either FORTRAN or MACRO-II (PDP-11 assembly language). DSL is clear and concise and allows great flexibility in designing displays. Displays can be defined in three ways: statically, not changing during the program; dynamically, using data from the execution of the program; or a combination of both ways. DSL provides for naming a field or group of fields so that they can be referenced by routines which modify, extract data from, or dynamically accentuate such named fields. This accentuation can be applied to any or all fields in the display, either when the display is defined or later during program execution.

There are 16 levels of accentuation, including the normal level (no accentuation), and a "zero" or blank level (data input is not displayed). The blank level is useful for entering passwords or other classified information, and for entering error messages into the display, which can then be used merely by dynamically changing the accentuation level.

DSL provides formatting capabilities, which include:

- Floating tab positions (up to 16) which are set
according to actual data length for the fields and the width of the target screen display area.

- Right and left justification around a tab position.
- Center text on a line.
- End page.
- New line.
- Justify text against the right margin.
- Fill the space between two fields with dots.

DSL allows the programmer to specify data attributes for input fields, such as data type and length. Data types include:

- String fields which consist of any character in the print set. Specific string attributes include alphabetic, numeric, blank, and special characters (all other printing characters). Any combination of the four is also allowed.
- Integer fields.
- Real fields in which scientific notation is also supported.
- Minimum-maximum field where the integer entered must be between two limits.
- Two character choice fields consisting of one position where the user must enter one of two specified characters.

Any of the above fields may also be specified as mandatorily requiring input as opposed to optionally requiring input. Input fields can also be accentuated.

DSL thus frees the programmer from varying considerations of screen size, display techniques, accentuation techniques, input methods, and required hardware protocol.

FUNCTIONAL FEATURES OF TTDL

Screen areas

The terminal display device can be divided into multiple independent regions, each known as a Screen Area (SA). Each SA can be treated as a separate display, allowing multiple programs to be run simultaneously on a single terminal. SAs can be defined in any realistic size.

Command line

Each terminal has a one-line command line. Input on this line is passed to a command interpreter which checks the command against a list of system commands. The command is then dispatched to the appropriate routine for implementation. Error messages, either syntax errors from the interpreter or execution errors from the service modules, are displayed on the command line at the end of the entered command. Current commands include running and aborting tasks, and display paging commands.

Select menu

A common programming technique is to provide the user with a list of choices, or selection menu, for the user to select from. TTDL provides a select construct in which the programmer merely lists all the choices in the menu. The prompt line (e.g., ENTER DATA
... is automatically generated by TTDL, a different prompt being available for each terminal type. This construct also allows the terminal user to make full use of any available input devices which the terminal supports. These may include light-pen, cross-hairs, function keys, or he may simply type in his choice. The choice made is echoed in the input field of the prompt line. The program sees only an integer denoting the choice selected.

**Screen formatting**

TTDL provides automatic display formatting by fitting the logical display to the target SA. The process includes breaking lines that are too long by backscanning for a blank and breaking at that point for a display-only field, or breaking exactly at the end of the line for an input field. The display is broken into pages if it cannot be made to fit on a single page. Efforts are made to fit the display on one page, such as ignoring “new line” codes for compression of selection menus. If the display must be paged, efforts are made to keep a logically connected field from being broken between pages by moving the entire field to the next page. Page information is also included at the bottom of each page, i.e., page number, whether there are more pages or not, and paging commands which can be entered by input devices—if they are available.

**Data checking**

TTDL provides automatic input editing to insure that data meets the restrictions assigned by the programmer. The user is not allowed to leave the current page until all errors are corrected. The number of errors is displayed on the bottom line of the SA. Fields requiring mandatory input must also be filled in correctly, although optional fields can be left blank. The fields are then marked to show the application program those that had data entered.

In the case of string fields, any spaces not filled in always appear as nulls to the programmer, no matter what character is used to reduce the limit of input fields on the terminal. Selection menu choices are checked for the validity of the total number of choices available.

**Control input**

Control sequences can be entered apart from normal data input. These sequences trigger control functions on the SA in which the programmer is working. These functions include:

- Command line—transfer to enter data on, and return from command line.
- Paging commands—page forward, page back, first page.
- Enter data—data is ready to be returned to the application program.
- Cursor control—move cursor up, down, right, left, home. Cursor only stops on unprotected areas.
- Tab—tab to next or prior input field, tab to next field containing incorrect data (flagged by data checking routines), tab to the first input field on the next line.
- Move to the next active SA.
- Erase input field.

Each function is declared valid or invalid and ignored pursuant to the status of the SA.

**Display library**

The user can store displays on disk through a display library capability, referring to them by a six-character name. Functions supported are store, retrieve, and delete. This capability frees the application program from the storage overhead of having the displays defined internally. Checks are made for duplicate names when storing. When retrieving displays, the actual display name is checked against the active displays; if it has the same name, it is renamed, and the new name is given to the program to use. This allows the same display to be used concurrently by different terminals controlled by the same program.

**TTDL SOFTWARE MODULES**

TTDL was designed using two kinds of software modules: primary modules, which drive the system; and support modules, which provides sub-processing and data support for the primary modules.

The primary modules are the Preprocessor, Postprocessor, Secondary Input/Output, and Primary Input/Output modules. They handle the actual transfer of data to and from the terminal.

- The preprocessor translates the DSL into a Terminal Independent Format (TIF).
- The postprocessor formats the data to fit the SA, and provides dynamic display paging support.
- The secondary input/output module translates common capabilities into terminal dependent functions.
- The primary input/output module transfers data to and from the terminal.

The support modules are Buffer Manager, Process Control, Field Routines, Application Interface, and Display Library modules.

- The buffer manager directs the storage and retrieval of TIF data.
- The process control module supports intra-task and inter-task communications.
Field routines allow modification to and retrieval of data from the display by the application program.

The application interface module provides a table-driven interface between the applications program and TTDL.

The display library module handles the storage and retrieval of displays from disk.

The following paragraphs provide a complete description of each TTDL software module.

**PRIMARY TTDL MODULES**

**Preprocessor (PRP)**

The preprocessor is a finite-state compiler which translates the DSL specified by the application program into a Terminal Independent Format (TIF). Using state tables, the preprocessor translates the data from DSL to TIF, which is suitable for storage on disk and recall for retrieval on any type of terminal. The preprocessor is completely terminal independent, requiring no information about the display terminal.

As a finite-state compiler, the preprocessor uses a series of macros to conduct state-translation processing. This construction allows easy additions to the language by merely making the additions to the state tables and combining them with the appropriate processing macros.

**Postprocessor (PSP)**

The postprocessor formats the TIF produced by the preprocessor to fit the target screen area. It builds a transaction file correlating data in the TIF to locations on the SA. Algorithms break lines that are too long, and automatically pages displays which are too big for one SA. The resulting transaction file is sent to the secondary I/O module for further processing. The postprocessor is virtually terminal independent, requiring only SA size and the number of display characters required for certain functions.

**Secondary I/O (SIO)**

The secondary I/O module produces the actual data stream sent to the terminal. As input, it uses the TIF, the transaction file built by the postprocessor, and a set of terminal dependent protocol tables and routines.

The secondary I/O is table-driven, a feature which facilitates addition of new terminals to the system. Addition of a new terminal requires only the addition of protocol tables and those routines which are unique to the terminal or new to the system. As more terminals are added to the system, the number of new routines decreases; most routines are common to a class of terminals.

Specific functional capabilities of SIO include: adherence to hardware protocol for each terminal; display field accentuation; addition of dots for the DSL feature; compensation for differences between terminal types; and hardware simulation with software routines.

**Primary I/O (PIO)**

The primary I/O module handles the actual transfer of data to and from the terminal. Inputs to primary I/O from the SIO are the address of the data and its length. Important features include: interruption of a read to allow a write to occur, and restoration of the read at the exact point of interruption; queuing of other reads while a read or write is in progress; validation of control sequences entered by the user—invalid sequences are ignored; support of zero level accentuation by echoing blanks to the terminal when data is entered; and asynchronous operation, by processing requests on other terminals until recently issued I/O is completed.

The primary I/O is also table driven, having tables of write routines, read routines, etc., which are dispatched by terminal type.

**SUPPORT MODULES**

**Buffer manager**

TTDL uses dynamic storage for its work areas, the majority of which are used for TIF. The buffer manager is responsible for handling of these buffers.

TTDL uses 512 byte blocks of dynamic storage allocated by the Intertask Coordination Module (ICM) of TOSS. Using a logical-to-physical mapping algorithm, the buffer manager translates TIF addresses into a specific block of TIF. The buffer manager pages the buffers to disk on a demand basis when in-core storage runs short, using an extensive algorithm to determine which blocks to write out. A currency concept is also used, through which a certain buffer is kept current for the TIF being processed, with a pointer to the current character in the buffer. Functions include: set currency; get current character; put a character; access a block; deaccess current block; and map into current location. This last function allows the requesting module to directly access the current block.

**Process control module**

The process control module provides the control required by TTDL to permit the simultaneous support of multiple tasks, multiple terminals and terminal types, and multiple screen areas on each terminal. The process control module assigns a Process Control Block (PCB) to each individual process. The PCB contains
all appropriate data for the process, including a stack
area where data may be stored if the process is sus-
pended. In addition to providing intra-task control,
inter-task communication is achieved by queueing
PCB's from one task to another.

The process control module also resolves resource
contention by suspending a process until the required
resource (such as nodes, buffers, or PCB's) is avail-
able.

Process control functions include: passing control
to another process; suspending a process; spawning a
ew new process; ending a process; and subprocess calling
and return (analogous to subroutine call and return).

These functions manipulate queues of process. A dis-
patcher module provides the control mechanism for
dispatching processes to be run and identifying the
task in which the process resides. The dispatcher also
handles Service Request Blocks (SRB) which are
requests for service issued by application programs
through ICM.

Field routines

Field routines consist of a group of modules which
allow the application programs to interact with the
TIF for their display. Individual modules include:
initialize a field to the null state; set fields to string,
integer, or real values depending on field type; retrieve
data from string, integer, or real fields; change accen-
tuation levels of display fields; test the availability of
input for input fields; and set origins for generated
field names. The largest modular group is a set of
routines to find specified fields in the TIF which the
application programs desire to access. Since the
processing performed by most routines is minimal,
addition of new routines is easily accomplished.

Application interface module (AIM)

The application interface module is the entry point
for all application calls. It checks the format of each
call and formats a service request block for trans-
mittance to the appropriate TTDL task. The AIM is
FORTRAN compatible, and addition of a new host
language would require only a small system enhance-
ment. TTDL task specifications would not be affected.

The AIM is table driven, so that a new call requires
only the addition of two macros. The AIM also pro-
vides conversion functions for dynamic display defi-
nition from data provided during execution.

Display library

The display library module consists of two parts.
One part performs the directory functions and alloca-
tion of disk storage. The second part, the external
processor, performs the formatting of the display TIF
and associated information for storage on disk.

A directory entry consists of a 6-character ASCII
name, the virtual block number of the display, and
the number of blocks allocated. Each directory has a
header which contains a pointer to the next directory
and the number of blocks allocated for a high block
number. Each directory is one disk block long. A
deleted directory entry is marked such that the blocks
may be reused.

While the display library keeps track of display
storage, the external processor formats the informa-
tion in the TIF and additional information required
when the display is reformatted. Parameters to the
external processor include a buffer to store the infor-
mation in and its length. The external processor can
also be used to send displays to and from other systems.

DATA AND CONTROL FLOW

The following paragraphs describe the typical flow
of data and processing to and from a terminal. This
flow is illustrated in Figures 2 and 3.

Display definition and output

Operations start with a call to the routine DISDEF
(display definition). DISDEF formats a request speci-
fying the argument and passes it to the preprocessor.
The preprocessor then maps into the application pro-
gram and retrieves the DSL. The DSL is then con-
verted into TIF and given to the buffer manager for
storage.

Once the display is defined (by one or more calls to
DISDEF) the program indicates its desire to output
the display via the FLASH call. This is dispatched to
the postprocessor which creates a transaction file based
on the TIF and the size of the target SA. This trans-
action file relates TIF data to SA position.

Control is then passed to the secondary I/O module,
which uses the TIF, the transaction file, and terminal
dependent tables and routines to create the output
stream. SIO also adds the hardware protocol for such
features as protected/unprotected fields, emphasis
(blinking, reverse video, etc.), code transliteration,
and cursor positioning.

Finally the buffer containing the output stream is
passed to the primary I/O module, which transmits it
to the appropriate terminal.

Data input

After the display has been output, the SIO then pre-
pares to input data. This data can consist of control
sequences or input for the program. The SIO sends a
request to the PIO for each field, stating how many
Figure 2—Input flow.

Figure 3—Output flow.
characters are required, any accentuation information, and buffer addresses for the input and output buffers. The PIO then returns control whenever the number of control characters or a valid control sequence is entered. The SIO then either goes to the next field, or processes the control sequence. When all the data is entered, the SIO returns control to the application program, which can then modify or retrieve the data entered.

SUMMARY

As can be seen from the above discussion, TTDL is a useful flexible language which can effectively facilitate communications among the many and various terminals of large data processing systems. Such a language delivers powerful capabilities to system users—capabilities which are necessary in the rapidly changing data processing environment.