1984

An Overview of the DASH Intelligent Terminal Project

John T. Korb
Purdue University, jtk@cs.purdue.edu

Report Number:
84-492

An Overview of the DASH Intelligent Terminal Project

John T. Korb

October 1984; Revised February 1985

CSD-TR-492

Department of Computer Science
Purdue University
West Lafayette, IN 47907

ABSTRACT

Named DASH, for Dynamic Access to Shared Hosts, the intelligent terminal portion of the TILDE research project is exploring ways to use microprocessor-based workstations to simplify and enhance user access to a network of mainframe computers. Unlike projects that use workstations as stand-alone processors or projects that use a collection of workstations to solve a single problem in a distributed fashion, DASH uses the workstation as a helper, or agent, to access network services and to interact on behalf of the user with programs running on one or more remote host processors.

The DASH prototype is based on SUN workstations running the Stanford V Kernel connected via an Ethernet network to a collection of VAX processors. While the prototype workstation is a rather powerful machine connected to multiple hosts via a high-speed network, the DASH design permits an environment in which a small personal computer is connected to a single host over a dial-up phone line or long-haul network connection.

---

* This work was supported in part by grants from the National Science Foundation (MCS-8219178), SUN Microsystems Incorporated, and Digital Equipment Corporation.

An Overview of the DASH Intelligent Terminal Project

1. Introduction

The advent of low-cost computing power, together with the development of high-speed networks, has spawned computing environments consisting of several medium-speed computers joined by networks. Research projects have undertaken different ways to do computing in this kind of environment [Fin80, Get84, LLA81, PWE83, SoF79].

The Tilde research project at Purdue [CKT84] is building a prototype distributed computing system that includes a set of machines joined by a high-speed local area network to form a *computing engine* with an interactive interface to the computing engine provided by an *intelligent terminal*.

Each individual machine in a computing engine provides a set of *services* such as electronic mail, general-purpose command interpretation, hardcopy printing, or high-speed vector processing. Heavily used services may be duplicated on several machines to improve performance. Less heavily used services, or services that require special-purpose hardware may be available on only one machine in each computing engine, or may be available only on remote computing engines.

Intelligent terminals are used to access the computing engine. Using the same hardware as personal computers or workstations, intelligent terminals are powerful enough to perform many tasks, such as editing, word processing, and simple numerical calculations. They promptly respond to input events, such as keystrokes and mouse actions, and can efficiently maintain a large-format bitmapped display. They are not powerful enough for tasks such as large database searches, or data processing for large businesses.

While microprocessors continue to get faster and smaller, their power for performing complex computations is still severely limited. If software systems remained the same size (or, since much work is being done to optimize existing systems, getting smaller) the explosive growth in power of workstations would seem to spell the end of large-scale timesharing mainframes. Instead, software systems are getting *larger* not smaller, and it is not clear that the improvements in workstation technology will be able to keep pace with this growth.

To the extent that users outgrow their workstations, and the size and complexity of software systems continue to grow, users will continue to depend on large, timesharing systems.

Timesharing is a cost-effective way to provide high-quality computing resources. The problem with timesharing is the limitation on the types of tasks for which it is suitable: CPU-bound computations with little real-time interaction. Programs that rely on servicing frequent interruptions from the user, such as screen-oriented editors, become unusable when the timesharing system is moderately loaded.

Even with a dedicated timesharing system (for example, a UNIX-based workstation), the sophisticated user soon has enough background tasks executing to bog down the response of interactive programs. In addition, interactive programs suffer from a variety of other problems.

- Most interactive programs (especially screen-oriented ones), have widely vary-

ing command and display organizations. It is difficult to switch back and forth between them. Similar operations, such as deleting an object, are not specified the same way.

- Taste varies widely among users and programmers. The merit or importance of a program is independent of its user interface. Too often, though, the interface is either so good or so bad (or, simply, so different), that it is the dominant feature.
- Interactive programs are hard to write. A consistent, short command set must be designed. A display update algorithm must be written. The screen layout must be designed. Errors must be displayed and handled reasonably.
- Interactive programs are not built incrementally or from smaller pieces, like tools [KeP76] that read from standard input and write to standard output. Instead, an interactive program represents a large, concerted effort that must be reproduced everytime a new application is implemented.

2. Related Work

How can these problems with interactive systems and timesharing be solved through the use of intelligent terminals? Many research projects have explored uses for intelligent terminals; most of these projects can be classified into one of two categories.

- *virtual terminal.* Escape sequences, for example, are transmitted to the terminal to cause cursor movement or to display graphics [Bar77, BaM78, Day80, MED79, ScD76, ScD78]. Many virtual terminal systems are designed for specific applications; for example, satellite graphics systems [ClD83, DiT75, Fol76, Gra79, KeM76, vSH74] and data entry systems [Naf78]. More powerful virtual terminal systems may multiplex multiple command interpreters (*shells*) on a single display [Ber79, Car83, MeM81, Pik83].
- *workstation.* The intelligent terminal is another processor in a (conceptually) homogenous system of processors [Bar79, ChZ83, GoR79, LNT84, SIK82, WaA79].

These two approaches suffer from opposite problems: Virtual terminals are coupled too tightly to the remote host, while workstations are coupled too loosely (if at all).

In the virtual terminal approach, the host views the virtual terminal as just another terminal with a different (usually higher-level) set of commands. The host programs must be written as interactive programs (with screen update algorithms and user command parsing). Responsiveness suffers if the host is busy with other users.

In the workstation approach, access to remote hosts is done explicitly by the user, using some form of remote job entry protocol. The interactiveness of the workstation is maintained, but the computing power of the workstation cannot be transparently expanded when the need arises. The user must consciously request to use one of the other processors in the system.

Both the virtual terminal approach and some of the more recent work using workstations to access remote hosts [LaN84] have the disadvantage that the communication between the virtual terminal/workstation and the remote host is at a low level. Typical commands to the virtual terminal still include picture drawing, menu manipulation, and cursor tracking.

An alternative approach is to use the intelligent terminal as a special-purpose *front-end processor* between the user and a network of host processors [And75, AnG76, Car82, LaR79, Lan80, MOK78, Miv76]. This approach, as it relates to the DASH project, is the subject of this report.

3. The DASH Approach

The goal of the DASH project (Dynamic Access to Shared Hosts) is to exploit the intelligent terminal as an agent that performs common tasks on behalf of the user (such as logging in to various machines on the network, or locating services), as well as providing a highly interactive front-end to non-interactive programs running on the computing engine. In this way, DASH forms a framework for writing interactive programs. The roles provided by the intelligent terminal can be divided into three categories: user agent, interactive front-end, and user-defined interface.

User Agent

The intelligent terminal requests service from individual machines of the computing engine on behalf of the user. It is responsible for locating special-purpose or lightly-loaded processors as well as handling the details of authorization (by logging in for the user) and resource requesting. This role is similar to the role provided by the V Executive (although authorization is not fully unified) [BBB84].

Interactive Front-End

The intelligent terminal provides limited local computing power for special-purpose interactive tasks (for example, editors). By handling all interactive processing such as keyboard and mouse input at the intelligent terminal, the host processors that make up the computing engine are relieved of the context switching burden required to implement highly interactive programs. This layering of resource utilization not only reduces the load on the computing engine, but also improves response to the user.

DASH communicates with the remote hosts at a higher, more abstract level than typical virtual terminal or workstation-based interactive systems. The interface provided by DASH does not include commands to pop up windows or draw menus, but rather commands that say "here are some commands that can be executed." Specifications provided by the user determine how these commands are to be displayed and selected for execution.

The DASH interface acts as a layer between the well-defined, relatively static computing engine interface and the user-defined, flexible intelligent terminal interface provided to the user.

The interface provided by the computing engine is command oriented. The system responds to line-oriented commands or command sequences. Input to a

command can come from the user, a file, or another command. Likewise, output from a command can be directed to the user, a file, or another command.

The interface provided by the intelligent terminal is interactive, asynchronous, and window oriented. The programmable portion of DASH acts as an interface between these two layers. This separation of responsibilities allows the highly interactive programs to be placed on the unloaded, low-cost, intelligent terminal; with communication between the intelligent terminal and the remote host via cost-effective network packets.

By maintaining a relatively high-level interface to the computing engines, the intelligent terminal allows programs that function as standard "UNIX tools," filters and pipes, to be easily written. The intelligent terminal can then provide an interactive, screen-oriented interface to these line-oriented programs. Users continue to use the tools approach to organize and structure their programs. They simply add another layer on top (perhaps later), that defines a convenient user interface. This separation, both logical and physical, of a process into interactive and noninteractive components is efficient, modular, and convenient for the user.

User-Defined Interface

DASH users customize the interactive interface to suit their own personal needs or tastes. Decisions about key bindings, window organization, command menus and invocation, error handling, and so on can be handled uniformly by the intelligent terminal. These decisions can also be customized to the user's preference, without modifying the programs that perform the bulk of the processing on the computing engine.

Emacs is an example of an interactive system that is both extensible, in that users can write their own functions that are essentially indistinguishable from built-in functions, and customizable, in that both built-in and user-defined functions can be bound by the user to arbitrary keys [Sta81]. Users have written large collections of functions, or "packages", that dramatically extend the built-in capabilities of the system.

In practice, however, because different users have written these packages, and because each package tends to specify its own key bindings, the typical Emacs user is faced with the problem of memorizing a variety of inconsistent key bindings. Furthermore, if the user changes a binding in one package, related bindings in other packages remain unchanged.

The problem is that, while the bindings are under control of the individual users, they are in fact made by the variety of users who write the packages. Since the bindings are not made by a single person, they are not consistent across all packages.

DASH avoids this problem by adding *generic commands* as an intermediate layer between key strokes and built-in (as well as, user-defined) functions. The user specifies the bindings of keystrokes to generic commands, while the implementation controls the bindings of generic commands to specific functions [Kri84].

For example, the DEL key may be bound to the generic function `delete-object`. An ordinary text package would contain a binding of `delete-object` to the built-in

function `delete-region`, while a mail system package would bind `delete-object` to the mail-system function `delete-message`. The user views the operations as similar enough that binding both of them to the same key creates no confusion, and simplifies memorization of key bindings.

4. Research Areas and the DASH Prototype

The first project will be to build an interactive interface that implements the goals described in this paper. This prototype will explore several research areas and provide a test bed for evaluating the intelligent terminal as a high-level network interface. Listed below are several goals and research areas to be examined by the DASH prototype.

1. Allow user control over all actions that affect the interface. For example, placement of windows by interactive programs should be through a user-controlled interface procedure. This procedure can determine window size and placement under user-specified guidelines, rather than by fiat or by explicit user interaction.
2. Never force the user to enter input at unexpected or unpredictable times. If input may be required, the user should be given a way to write a procedure to provide it. For example, if an unusual condition occurs the interface should make an attempt to call a user-defined procedure to handle the exception, rather than requesting a response from the user.
3. Allow a user to sit down at another user's session and change (temporarily) the bindings and other attributes of the user interface. This feature should allow users to work more easily together, although they will still be speaking a different command set.
4. Investigate programs that follow the "editor paradigm" in which all operations can be characterized as editing operations. For example, an editor-like interface has been used to implement Unix inode editing, machine state changes, and interactive graphics [Fra80]. Other programs include mail box editing, directory editing, and core image editing (for example, with a source-level debugger).
5. Consider the design of two orthogonal interfaces: one between the remote host and the workstation database (the *data interface*), and one between the workstation database and the display (the *display interface*). The data interface might be organized using boxes and glue as in TEX [Knu84]. The display interface would allow the user to specify window sizes and locations, and the system would then do "page layout" to fit the data into these windows.
6. Investigate pointing-device (mouse) techniques. There are several paradigms for using the mouse to invoke commands. Some examples are
   a. high-lighted "buttons" on the screen,
   b. pop-up menus,
   c. dedicated screen regions (for example, scroll bars), and
   d. dedicated windows that can be scrolled, enlarged, and reduced.

   In any event, the actual choice of a technique under user control, not program (host) control. Thus, the data interface (in the point above) should

include the notion of "commands" and command names, but not how those commands are to be displayed to or invoked by the user.

7. DASH will use Icon [GrG84] as its extension language [Mit84].
8. Design a shell that runs on the intelligent terminal. There are two common models: (1) one window per host login session (typically, with a window emulating a standard terminal type) [SSS84], and (2) one window per "work session," with each work session accessing one or more hosts [BBB84]. Another proposal might be called "multiple windows per host login session" or even "multiple windows per work session."

   Yet another idea might be to have two windows: one standard window (as in item 1) and another "history window" that contains all the typed commands so that previous commands can be easily selected and modified. Other nice features include the ability to distinguish (visually and by a program) typed input from process output (for example, by different colors or shades).

   Perhaps each command and its output could automatically be put into a separate window, and execute in "background." The sequential nature of many commands sequences (that is, the requirement that one command complete before the next is begun) needs to be maintained, but maybe not by the simple single-input stream method used now.
9. Consider alternative intelligent terminal hardware, such as personal computers, as well as host/terminal connections over low-speed serial lines or long-haul networks.

5. Schedule

A graduate seminar investigated a variety of user interfaces and approaches to interactive systems design in the Fall semester of 1984. The results of this study are currently (Spring 1985) being applied to the development of a DASH prototype based on SUN workstations running the V-Kernel.[VSystem Reference Manual 1984.] The prototype is expected to be functional and an evaluation completed during the Summer of 1985.

References

[And75] R. H. Anderson, Advanced intelligent terminals as a user's network interface, *IEEE COMPCON Proceedings*, Fall 1975, 180–182.

[AnG76] R. H. Anderson and J. J. Gillogly, The RAND intelligent terminal agent (RITA) as a network access aid, *Proceedings of the AFIPS 1976 NCC* **45**, (1976) 501–509.

[Bar77] D. L. A. Barber, A virtual terminal protocol based on the use of zones, *Computer Communications Review* **7**, 3 (July 1977) 59–76.

[Bar79] D. L. A. Barber, Protocols for intelligent terminals, *Computer Communications Review* **9**, 3 (July 1979) 13–22.

[BaM78] E. Bauwens and F. Magnee, The virtual terminal approach in the Belgian University Network, *Computer Networks* **2**, (1978) 297–311.

[BBB84] E. J. Berglund, P. Bothner, K. P. Brooks, D. R. Cheriton, S. E. Deering, J. C. Dunwoody, R. S. Finlayson, D. R. Kaelbling, K. A. Lantz, T. P. Mann, R. J. Nagler, W. I. Nowicki, P. J. Roy, M. M. Theimer, and W. E. Zwaenepoel, V-System 5.0 Reference Manual, October 18, 1984.

[Ber79] T. S. Berk, A satellite graphics system for error-prone users, *Computers and Graphics* **4**, (1979) 185–187.

[Car82] T. T. Carey, A workstation for interaction styles, *IEEE COMPSAC Proceedings*, 1982, 303–307.

[Car83] T. A. Cargill, The Blit debugger (preliminary draft), *Software Engineering Notes* **8**, 4 (August 1983) 190–200.

[ChZ83] D. R. Cheriton and W. Zwaenepoel, The distributed V kernel and its performance for diskless workstations, *ACM SIGOPS* **17**, 5 (June 1983) 129–140.

[ClD83] J. H. Clark and T. Davis, Work station unites real-time graphics with UNIX, Ethernet, *Electronics* **56**, 21 (October 20, 1983) 113–119.

[CKT84] D. Comer, J. T. Korb, W. Tichy, and T. Murtagh, The TILDE project, Computer Science Dept. Tech. Rep. 500, Purdue University, November 23, 1984.

[Day80] J. D. Day, Terminal protocols, *IEEE Transactions on Communications* **COM–28**, 4 (April 1980) 585–593.

[DiT75] J. C. Dill and J. J. Thomas, On the organization of a remote low cost intelligent graphics terminal, *Computer Graphics* **9**, 1 (June 1975) 1–8.

[Fin80] R. Finkel, The Arachne kernel, Tech. Rep. TR–380, Univ. of Wisconsin, Apr. 1980.

[Fol76] J. D. Foley, A tutorial on satellite graphics systems, *Computer* **9**, 8 (August 1976) 14–21.

[Fra80] C. W. Fraser, A generalized editor, *Comm. ACM* **23**, 3 (March 1980) 154–158.

[Get84] J. Gettys, Project Athena, *Proc. of the Summer USENIX Conf.*, June 1984, 72–77.

[GoR79] A. Goldberg and D. Robson, A metaphor for user interface design, *Proceedings of the 12th Hawaii International Conference on System Sci-*

ences, 1979, 148–157.

[Gra79] Graphics Standards Committee, Distributed graphics systems — draft report, *Computer Graphics* **13**, 3 (August 1979) V1–V10.

[GrG84] R. E. Griswold and M. T. Griswold, *The ICON Programming Language*, Prentice-Hall, 1984.

[KeM76] R. G. Kellner and L. D. Maas, A developmental system for microcomputer based intelligent graphics terminals, *Computer Graphics* **10**, 2 (July 1976) 139–142.

[KeP76] B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison–Wesley, 1976.

[Knu84] D. E. Knuth, *The TEXbook*, Addison-Wesley, 1984.

[Kri84] B. Krishnamurthy, A uniform generic command interface: the DASH approach, Computer Science Dept. Tech. Rep., Purdue University, December 1984.

[LaR79] K. A. Lantz and R. F. Rashid, Virtual terminal management in a multiple process environment, December 1979.

[Lan80] K. Lantz, Command interaction in distributed systems, *IEEE COMPCON*, Fall 1980, 25–32.

[LNT84] K. A. Lantz, W. I. Nowicki, and M. M. Theimer, Factors affecting the performance of distributed applications, *Computer Communication Review* **14**, 2 (June 1984) 117–123.

[LaN84] K. A. Lantz and W. I. Nowicki, Structured graphics for distributed systems, *ACM Transactions on Graphics*, January 1984, 23–51.

[LLA81] E. D. Lazowska, H. M. Levy, G. T. Almes, M. J. Fischer, R. J. Fowler, and S. C. Vestal, The architecture of the Eden system, *Proc. of the 8th Symp. on Operating System Prin.*, Dec. 1981, 148–159.

[MED79] F. N. Magnee, A. Endrizzi, and J. Day, A survey of terminal protocols, *Computer Networks* **3**, (1979) 299–314.

[MOK78] J. M. McCrossin, R. P. O'Hara, and L. R. Koster, A time-sharing display terminal session handler, *IBM Systems Journal* **17**, 3 (1978) 260–275.

[MeM81] N. Meyrowitz and M. Moser, BRUWIN: An adaptable design strategy for window manager/virtual terminal systems, *ACM SIGOPS* **15**, 5 (December 1981) 180–189.

[Miv76] J. Michel and A. van Dam, Experience with distributed processing on a host/satellite graphics system, *Computer Graphics* **10**, 2 (July 1976) 190–195.

[Mit84] W. H. Mitchell, An Icon subsystem for UNIX Emacs, TR 84–8, The University of Arizona, Department of Computer Science, May 1984.

[Naf78] N. Naffah, High level protocol for alphanumeric data-entry terminals, *Computer Networks* **2**, 2 (May 1978) 84–94.

[Pik83] R. Pike, Graphics in Overlapping Bitmap Layers, *Computer Graphics* **17**, 3 (July 1983) 331–356.

[PWE83] G. Popek, B. Walker, R. English, C. Kline, and G. Thiel, The LOCUS Distributed Operating System, *Operating Systems Review* **17**, 5 (Oct.

1983) 49–70.

[ScD76] P. Schicker and A. Duenki, Virtual terminal definition and protocol, *ACM SIGCOMM* **6**, 4 (October 1976) 1–18.

[ScD78] P. Schicker and A. Duenki, The virtual terminal definition, *Computer Networks* **2**, 6 (December 1978) 429–441.

[SIK82] D. C. Smith, C. Irby, R. Kimball, and E. Harslem, The Star user interface: An overview, *Proceedings of the AFIPS 1982 NCC* **51**, (1982) 515–528.

[SoF79] M. H. Solomon and R. A. Finkel, The Roscoe operating system, *Proc. of the 7th Symp. on Operating System Prin.*, 1979, 108–114.

[Sta81] R. M. Stallman, EMACS: The extensible, customizable self-documenting display editor, *SIGPLAN Notices Notices* **16**, 6 (June 1981) 147–156.

[SSS84] Programmer's Reference Manual for Suntools: the Sun Window System, January 1984.

[vSH74] A. van Dam, G. M. Stabler, and R. J. Harrington, Intelligent satellites for interactive graphics, *Proceedings of the IEEE* **62**, 4 (April 1974) 483–492.

[WaA79] S. K. Warren and D. Abbe, Rosetta Smalltalk: A conversational, extensible microcomputer language, *Proceedings of the 2nd Symposium ACM SIGSMALL* **5**, 2 (April 1979) 36–45.