

An Overview of the BTRON/286 Specification

While individuals may find the 32-/64-bit TRON Project's systems too costly, this 16-bit version should be more affordable and fill business data and graphics needs.

Ken Sakamura
University of Tokyo

Yoshiaki Kushiki
Matsushita Electric
Industrial Co., Ltd.

Kazuhiro Oda
Toshiba Corporation

BTRON, or Business TRON, is a set of operating system specifications for workstations and personal computers that employ bitmapped displays. The primary goals set for BTRON call for the realization of easy operation through a standardized human-machine interface (HMI) and the assurance of data compatibility across different applications and machines. Second, BTRON establishes a method for exchanging data and control across applications. Third, BTRON realizes a document management system in which text and graphics can be mixed freely and layouts handled with ease. And fourth, it offers standard support of English, Japanese, and other languages, providing a wide variety of fonts.

The BTRON specification incorporates an innovative information management model known as the *real/virtual object model*.¹ Other new concepts include BTRON's use of tags called *fusen* to define relations between applications and data. HMI specifications extend from keyboard and pointing-device design to the movement of objects on the screen and basic editors for both text and graphics. The standardized HMI, data compatibility, and other aspects take form in a consistent design approach followed throughout the entire TRON-project architecture.

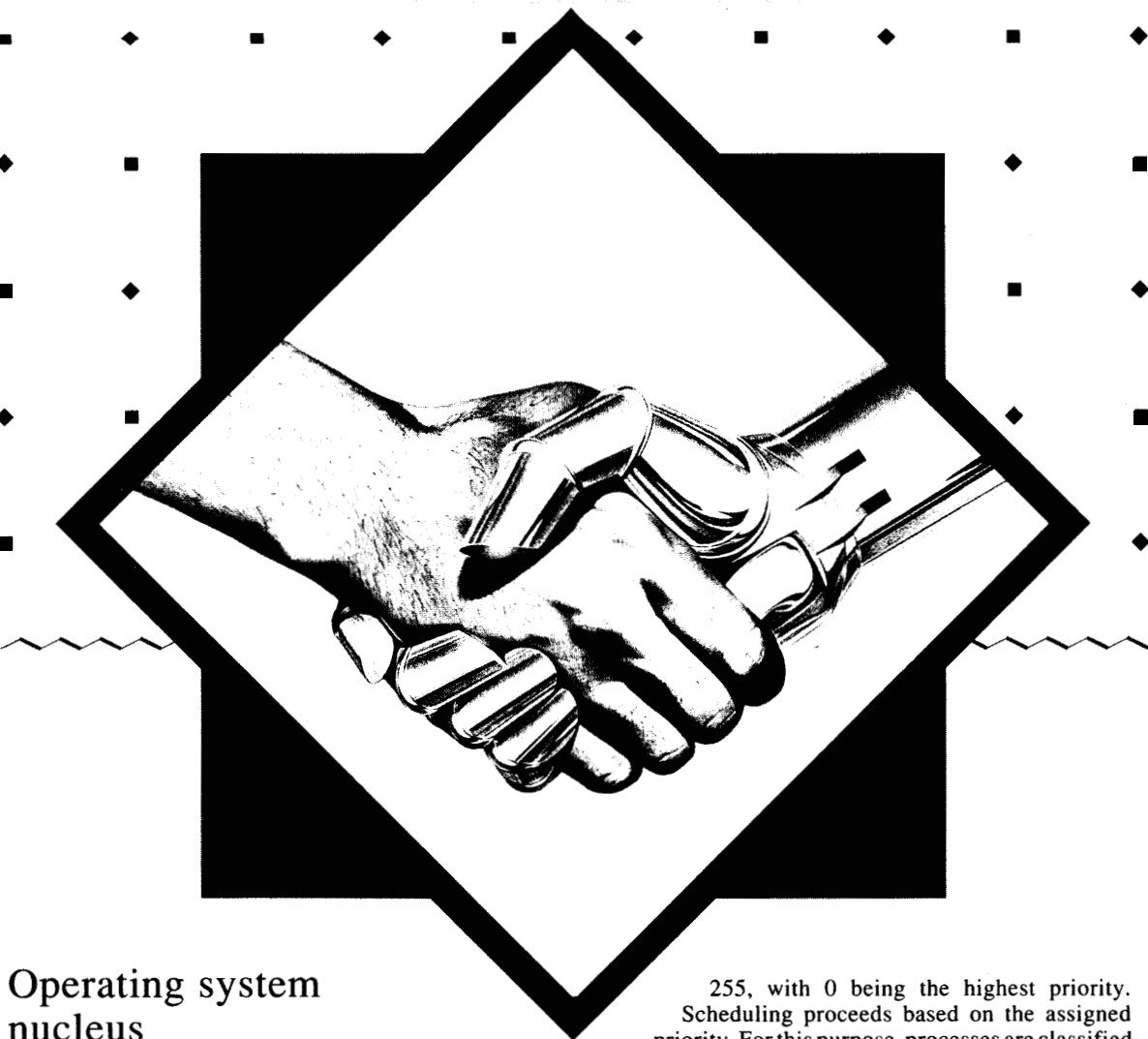
The BTRON specification achieves maximum efficiency when it is implemented on a family of processors designed as part of the TRON project: the VLSI CPU. The clearly hierarchical operating system design also assures ease of implementation when the nucleus is realized on other processors. Here we explain some details about this nucleus and the extended nucleus of the BTRON specification. We concentrate on the BTRON/286 specification, which is suitable for currently available 16-bit microprocessors. (See the BTRON/286 hardware and video image handling boxes on pp. 16 and 17.) Additional features, not found in the BTRON/286, will be added to BTRON/TRON VLSI CPU specification. Refer to Sakamura¹⁻³ for a general overview of BTRON/286 plans.

Software configuration

The BTRON specification defines a single-user, multiprocess operating system designed for a superior human-machine interface. It consists of the following three broad levels:

- the operating system nucleus,
- the extended nucleus, and
- system applications.

Figure 1 on p. 16 outlines the BTRON software configuration.



Operating system nucleus

The operating system nucleus provides basic services such as process management, interprocess communication, synchronization, and a systemwide naming space. Let's look at some of the features.

Process management. Processes are discrete units of the overall processing involved in program execution. A process ID (>0), given to a process when it is created, distinguishes the different processes. Processes have four basic types of status (see Figure 2 on p. 17):

- *Nonexistent.* The process has not been created.
- *Ready.* The process is executable and waiting to be dispatched.
- *Running.* The process is executing.
- *Waiting.* The process is waiting for a message, the passage of an indicated amount of time, input/output, and so on.

Processes go through the following transitions from one status to another, as a result of system calls or scheduling (dispatching, preempting).

Process priority and scheduling. When a process is created, it receives a priority numbered between 0 and

255, with 0 being the highest priority. Scheduling proceeds based on the assigned priority. For this purpose, processes are classified into three groups: absolute priority (priority between 0 and 127), round-robin 1 (priority between 128 and 191), and round-robin 2 (priority between 192 and 255). Scheduling takes place as follows:

- 1) If there are processes in the absolute priority group on ready status, the process with the highest priority changes to running status and executes. Otherwise, scheduling moves to step 2.
- 2) If there are processes in round-robin group 1 on ready status, scheduling assumes relative priority (explained later). The selected process (not necessarily the one with the highest priority) changes to running status and executes. Otherwise, scheduling moves to step 3.
- 3) If there are processes in round-robin group 2 on ready status, scheduling assumes relative priority. The selected process (not necessarily the one with the highest priority) changes to running status and executes. Otherwise, scheduling starts over from step 1 above.

Relative priority is a dynamic priority that is calculated using the statically assigned priority, CPU time usage of the process, and frequency with which the process has been scheduled. Relative priority keeps a

Video Image Handling on the BTRON/286 Implementation

The BTRON/286 hardware uses an optional Video Processor Unit to handle video image data. The VPU permits external video signals to be displayed on the screen of a personal computer, then digitized and stored into internal frame memory. Users can enlarge or shrink the image as well as repeat small images (tiling). Also, the digitized images can be shown superimposed on the computer screen. Other operations such as inversion and logical operations on the digitized images are available. With this VPU users can manipulate animated images taken from a videodisc or videocassette recorder system.

The extended nucleus of the BTRON/286 operating system contains a built-in software module called the video manager. This module isolates the applica-

tion programs from the details of the video hardware and offers a set of logical interface functions to access the video images. This isolation makes it very easy to develop application programs for handling many digitized images. Figure B is an example of an application that uses the video manager on the BTRON/286.



Figure B. Video, graphics, and text windows opened simultaneously on the BTRON/286 screen.

low-priority process from waiting forever.

Normally, system processes and real-time processes belong to the absolute priority group, while ordinary application processes belong to the other two groups. A process in the absolute priority group (priority between 0 and 127) can lock or unlock itself and can become resident in memory by locking the memory space of the process in memory. Such a process is said to be on lock status. The priority of a process once it has been created can be changed only within the same group; it cannot be shifted to a different priority group.

Message communication among processes. Each process contains a message queue in which messages are communicated among processes. A message moves to the message queue belonging to a process according to the destination indicated in its process ID. See Figure 3. The source is likewise identified by means of the process ID in the sending process. A message queue has a FIFO (first in, first out) structure, so that messages always enter the queue in the order in which they are sent. If the message queue at the destination is full when a message is sent, the sending process receives an error code that tells it to wait until space becomes available. Certain types of messages can be received selectively by specifying their types.

Synchronization and control among processes. Counting semaphores act as mechanisms for synchronization among processes and for mutually exclusive usage of shared resources. A semaphore ID assigned at

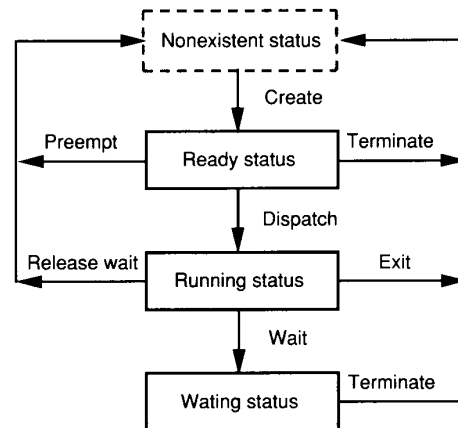


Figure 2. Basic status transitions of processes.

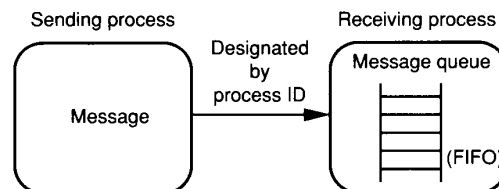


Figure 3. Message communication among processes.

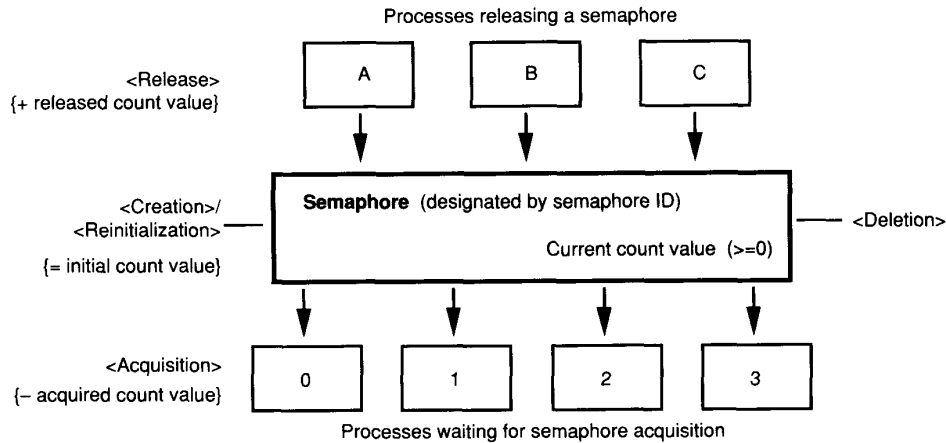


Figure 4. A semaphore operation.

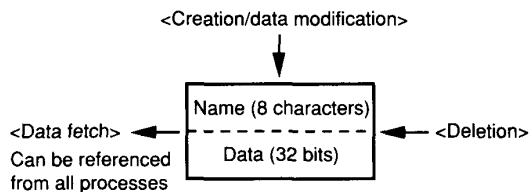


Figure 5. A global name data operation.

the time of creation identifies the dynamically created semaphores. Once a semaphore has been created, it can be used by every process. Figure 4 shows the semaphore operation.

Global name data management. Global name data can be defined as data with a globally visible name. Global name data can be shared among processes so they can exchange a small amount of data with high speed. Such data contain freely chosen names up to eight characters long by which all processes can reference and modify the data, as seen in Figure 5. The two kinds of global name data include one that continues to exist regardless of the existence of the process creating or modifying it and that must explicitly be deleted when no longer needed. (In BTRON/286, the modification of the global data is treated as if it were a creation of new data.) A second kind is automatically deleted when the process creating or modifying it is terminated. Either kind can be designated when creating or modifying global name data. Global name data mainly allow certain data to be referenced by any process.

Memory management. The BTRON specification provides for management of local and shared memory.

Local memory. This type of memory can be used in only one process and cannot be accessed directly by other processes. Local memory is created automatically at the same time a process is created and is released automatically along with the process at the time the process terminates. Application processes can acquire and use memory blocks of the required size from local memory. The acquired memory block contains continuous (logical) addresses, and the start (logical) address is returned to the application.

Shared memory. This type of memory can be used by all processes for data that are shared by several processes, for data used by the extended nucleus, and for other purposes. A number of memory pools make up a shared-memory area, though at system start-up, only the system memory pool exists. Application processes acquire shared-memory blocks of the specified size from the specified memory pool, a special memory pool consisting of the entire available system memory. Processes dynamically create ordinary memory pools that are identified by a memory pool ID given at the time of creation. Creation of a memory pool amounts to allocating part of the system memory pool and making it a separate, discrete memory pool. See Figure 6.

File management. The file structure adopted in the BTRON specification maps the real/virtual object model for the actual hardware. BTRON/286 users will manipulate the data based on this model, but the underlying data structure is implemented as files, as in conventional systems. Note that

- Files are structured as record streams; that is, they consist of ordered series of variable-length records.
- A random network of reference relationships exists among files based on links (virtual objects) included in files. (No directory exists as in conventional file systems.)
- These links (virtual objects) access files directly.
- Access is controlled via a user access level number (0-15) that is assigned to each user and a file access level number (0-15) that is assigned to each file.

Files and links. As just mentioned, a *file* is a stream of ordered, variable-length records that corresponds to a real object. A *link*, corresponding to a virtual object, points to files and serves as a clue for file reference. A link can be embedded in any file as a record, and more than one link can point to the same file. In this way an overall network of reference relationships is defined. Links indirectly reference files, and as a result the name of a file does not have an absolute meaning to uniquely identify a file. Rather, it functions as one search key. Any file name of up to 20 characters can be assigned, and the same name can be assigned to more than one file if desired. See Figure 7.

File structure. Every file system has one root file. By tracing the links included in the root file, users can reach all files in the file system, as shown in Figure 8. In the real/virtual object model, a root file corresponds to a device's real object. When a file system is created, it acquires a file system name up to 20 characters long and a device location name. The system and the users need the file system name, and the name of the root file, for absolute identification of the file system. The device location name, also up to 20 characters in length, indicates the physical device in which the file system is stored.

Attaching the file system. At system start-up, no file system exists. Before one can be used, an attachment operation as shown in Figure 9 must take place, whereby the name of the logical device in which the file system exists and the attaching name are specified. Detaching an already attached file system is done by specifying the name of the logical device to the detaching command. As a result, one cannot access a file via links that reference the files in the detached file system. The links are said to be on detached status. In the real/virtual object model, a link on detached status corresponds to a shadow object.

File ID. A file ID is a unique 16-bit number assigned to all files in the file system. The file ID of the root file in a file system is always 0.

Direct and indirect links. A fixed link, a link stored in a file as one record, can reference only files in the same file system. A link file is a special file to control

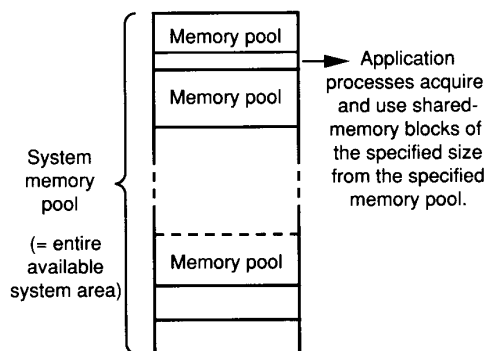


Figure 6. System memory pool.

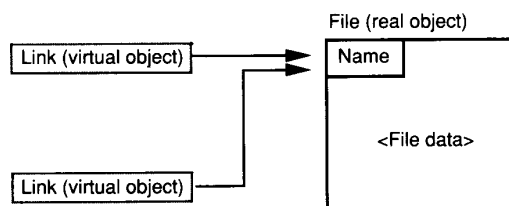


Figure 7. Files and links.

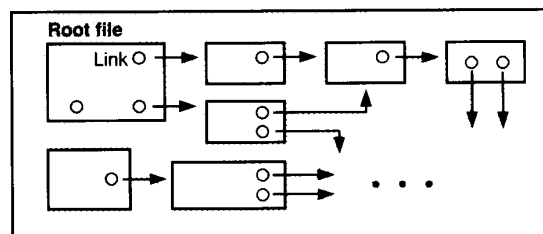


Figure 8. File system structure.

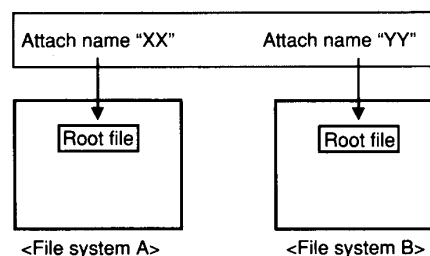


Figure 9. A file system attachment.

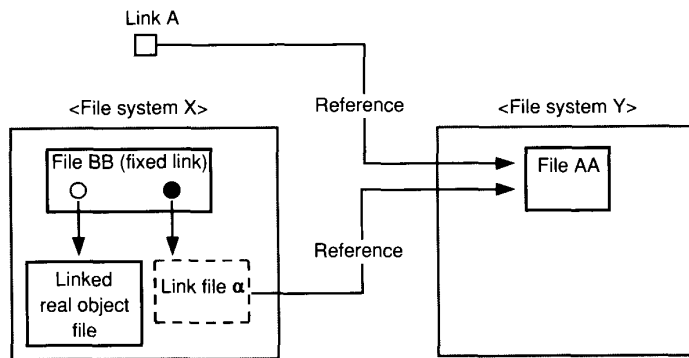


Figure 10. An indirect link and a link file. To store link A in file BB, first create link file α then store indirect link A' to α in file BB.

a reference to different file systems for configuring an indirect link. References to files in different file systems must be made via a link file in the originating file system. Links to link files are called indirect links because they add another level of indirection. As can be seen in Figure 10, references to files in the same file system can be made via a direct link stored in a file. An indirect link is a fixed link to a link file; a direct link is a fixed link to an ordinary file.

Event management. This capability uniformly treats keyboard and pointing-device operations as *events* by which users can interact with the computer. See Figure 11. These operations are recorded as one event after another in the systemwide event queue. Applications fetch these events one after another from the event queue, with the corresponding action being carried out in an event-driven form. This arrangement

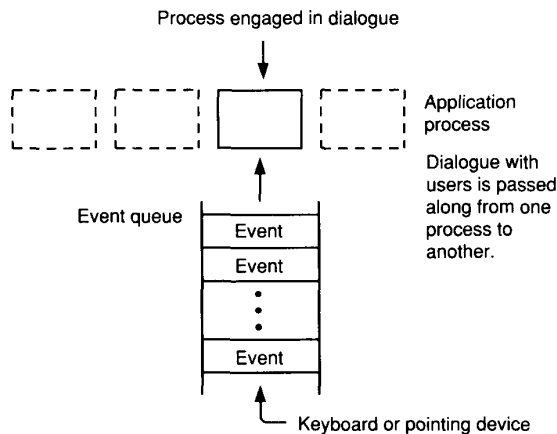


Figure 11. Event management. At a given time, only the process engaged in dialogue fetches an event, making use of the event-management capability.

is based on the rule that, at a given point in time, only the process engaged in dialogue uses the event management capability to fetch events. The BTRON/286 specification defines the following types of events:

- button down,
- button up,
- key down,
- key up,
- automatic repeat key,
- device,
- null, and
- application events 1 to 8.

Each event is associated with a bit mask, which identifies the event a process has fetched.

Device management. Device management functions include an application interface for the uniform handling of various devices, device-driver registration and management for various devices, and a device-driver interface. See Figure 12. The file management and event management capabilities in the operating system nucleus perform the actual device operations. A logical device name is a unique character string registered in the system and consisting of class, unit, and subunit elements. A *class* name indicates the type of device, such as *hd* for hard disk, *fd* for flexible disk, and so on. A *unit* distinguishes individual devices when a number of units of the same class exist. Normally, a single letter of the alphabet, starting from *a*, identifies units, for example, *hda0*, *hda1* for hard disk *hda*. When one unit is logically subdivided, a *subunit* distinguishes each part using numbers from 0 to 254 assigned in order.

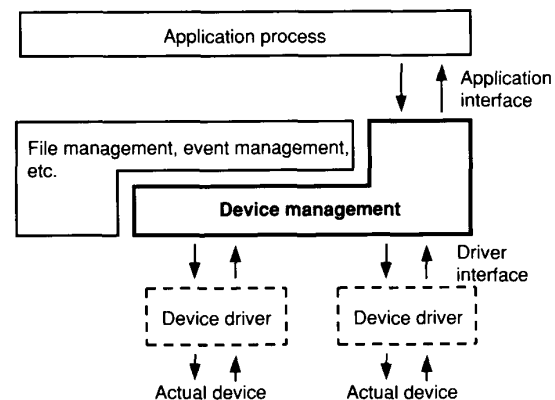


Figure 12. Device management.

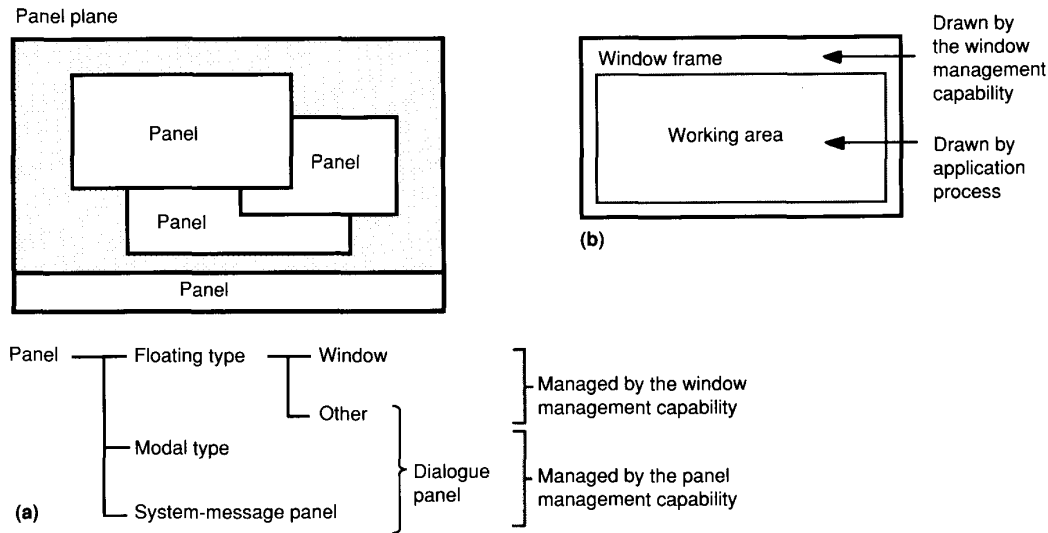


Figure 13. Panel (a) and window (b) management. The panel plane is the operation environment space on the display screen.

Extended nucleus

As shown earlier in Figure 1, the software contains many functions. Here, we explain three special features of the BTRON operating system:

- panel management,
- real and virtual object management, and
- TAD (the TRON Application Databus).

Panel management. The BTRON specification designates a rectangular region on the (virtual/real) display screen as a *panel*. A panel can be placed in an overlapping manner any place on the screen and displayed via programs. When a hidden panel is exposed (say, if a panel that obscures another panel is moved), the newly exposed area in the panel is redrawn to show the data that should be visible on the screen. The display of each panel is independent of what is shown in other panels.

When different panels appear on the screen, only one panel can accept a user's keyboard input. The user must click the button on the pointing device (a penlike accessory) to change the status of the panels so that a different panel can accept input. The BTRON specification makes it possible to realize a panel (windowing) system with a minimum of hardware and memory. A basic assumption is that no special hardware is supported.

Panel types. Panels can be classified into modal, floating, and system-message categories, as shown in Figure 13. Users call up *modal* panels on the screen via application programs; these panels can't be moved once they are displayed. Modal panels are always exposed unless hidden by other modal panels. When a modal panel is displayed, it is the only panel with which a user can interact.

A user can relocate *floating* panels by dragging the pointing device over the screen. When a floating panel can accept input, the user can choose to click the pointing device and access another panel so it will accept input. A floating panel that shows the content of a real object and allows users to interact is called a *window* in BTRON terminology. Usually, a window shows a part of a real object, and the user scrolls the screen to take a look at different parts of the object. Floating and modal panels that are not windows are called *dialogue* panels. Dialogue panels often offer the user an interface for setting application parameters.

The *system-message* panel is a long, narrow panel that appears at the bottom of the display screen. Only one system-message panel can appear on a screen. It holds system messages and is never hidden.

Windows can be extended to occupy the full screen (save, of course, the system-message panel) and are said to be in full-screen mode. The user can easily change a window so that it occupies either a small region or a full screen.

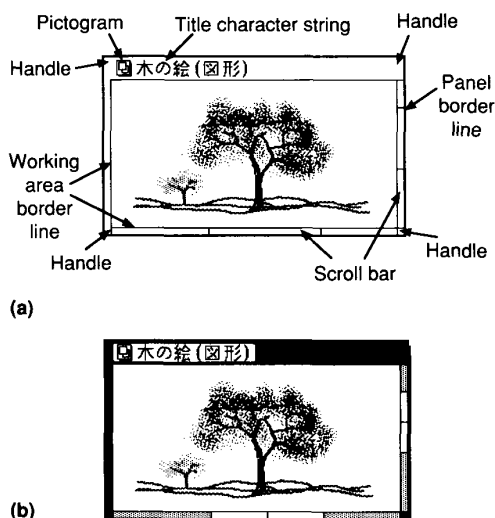


Figure 14. Standard window shape and title bar display (a) and a window in the input enable status (b).

Window display. Figure 14 shows the standard window shape and standard title bar display.

Window operations. Please recall that a window is a type of panel. There are five window operations:

- *Creating a window.* The user executes an application using a menu and clicks twice on a pictogram so that the selected virtual object is opened in the window.
- *Switching input enable status.* The user either clicks on the inside of the window with the pointing-device button or selects the real object name from the display management menu. (This allows the window to move to input enable status and become the first front window.)
- *Moving a window.* The user drags the pointing device over any part of the window frame other than a handle or scroll bars.
- *Resizing a window.* The user drags a handle inside the window frame to choose an arbitrary window size or a full-screen window. For a full-screen window, the user either selects the full-screen mode from the menu or clicks twice on the handle. Users can switch between the full-screen mode and normal (original) size easily.
- *Erasing a window.* The user either selects Quit from the menu or clicks twice on a pictogram.

Pointer shape. The screen displays a pointer shape (in the form of a hand for easy understanding; see

Figure 15), which indicates the active operation. When neither a pointing-device button, menu button, nor Command key is pressed, the pointer shape depends on the pointing-device position and automatically changes to a shape that indicates the operation possible in that position. (On a standard TRON digitizing pen, the pointing-device button is located at the tip of the pen and the menu button in the middle of its side. The Command key appears on the TRON keyboard.)

Real/virtual object management. These functions let the user display and manipulate real and virtual objects, as well as register and delete application programs. They also allow those application programs making use of real/virtual object management to display and manipulate real and virtual objects. In addition, all application programs fundamentally are executed via real/virtual object management. The real/virtual object management functions include:

- various functions for display and manipulation of virtual objects,
- various functions for display and manipulation of fuses,
- application program management (registration/deletion/execution),
- menu setting,
- file system-mounting management, and
- other application functions.

Virtual object operations. Listed here are the operations possible with regard to virtual objects. Users activate these operations directly on the screen either by using a pointing device or by selecting an item from a virtual object operation menu. The operations are

- selection,
- removing a disk,
- copy or movement,
- resizing,
- opening,
- closing,
- modification of a real object's name,
- modification of a relation,
- creation of a new revision,
- display of virtual/real object management information,
- opening as a window,
- modification of display attributes, and
- registration or deletion of an application.

Physical implementation. These objects are mapped to physical data that consist of many types of records. For example, a real object can consist of records that contain application programs, program-specific data, and TAD (explained next) format data that can be shared between applications and many systems. Each record of these objects contains subdivisions called segments.

TAD (TRON Application Databus). The BTRON specification provides a uniform data format called TAD, which permits the ready exchange of data among different applications. A TAD record makes up a part of the real objects in BTRON.

From the standpoint of assuring data compatibility between applications, TAD can be seen to have these four features:

- guaranteed compatibility of basic data (text, figure, and sound) across applications;
- variable-length segment format for easy data-exchange processing;
- the capability to hold application-dependent data; and
- a realization of the real/virtual object model in exchanged data, allowing external data referencing.

The fusen, handling application-specific data in TAD. TAD specifications are designed to guarantee data compatibility across applications at a minimum level. The TAD format is structured of text and figures as basic data, based on the notion that text and graphics are two kinds of data that can be handled by any application. However, if we define data format for text and figures in a very restricted way, many programs may suffer from the lack of flexibility and loss of efficiency. To avoid this problem, TAD permits an application-specific data format and clearly separates the application-specific data and application-independent data.

Application-specific data, such as a database index or a special binary data structure, are stored as *fusen* (a Japanese word for a small piece of paper used as a memo pad). Fusen is a generic name for application-specific segments. Although the content of a fusen is application-specific, its external format follows a certain rule. Among others, the application name that can interpret such fusen is stored in a defined format at the beginning of each such fusen segment. So when an application encounters a fusen segment, it can safely ignore the segment since it knows the length of the segment. In a sense, BTRON standardizes the way nonstandard data is stored.

Fusen can be stored like virtual objects inside a real object and are displayed as rectangular regions on the display. However, unlike virtual objects, each fusen itself holds some data internally. (A virtual object is a link to other real objects.)

Usually the user can change the content of a fusen by showing it on the screen and then invoking the associated application by clicking to have a dialogue panel appear. The user actually changes fusen by interacting through the dialogue panel.

Fusen can be classified broadly into the following types:

- *Functional.* This type of fusen holds data necessary for invoking application programs. If a functional fusen

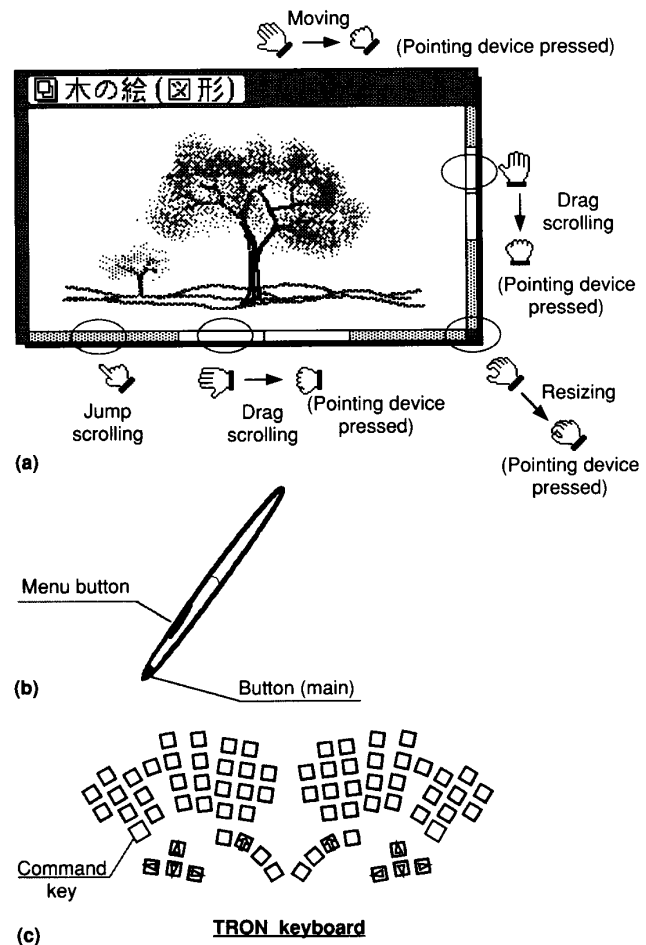


Figure 15. The pointer shape interacting with the screen (a), the pointing device (b), and the TRON keyboard (c).

is embedded in a real object, the fusen can specify the application that works on the real object.

- *System setting.* This type of fusen modifies system settings such as the volume of a beeping sound, which repeats automatically whenever a key is held down for a specific length of time.

- *Adjective.* This fusen is embedded in the text or figure record of a real object. An application will use the adjective fusen to interpret data with added detail. For example, font information or format information for a character is stored as adjective fusen in a TAD format data. The basic text editor, which is available on any BTRON-based system, can interpret the adjective fusen and act accordingly.

The following manipulations can be performed on fusen:

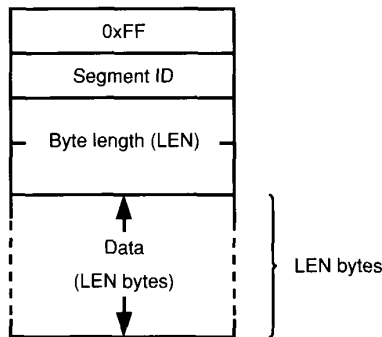


Figure 16. The structure of a variable-length segment.

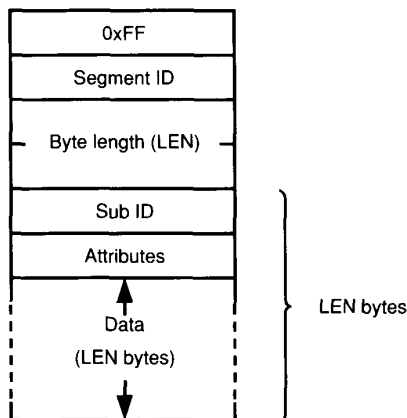


Figure 17. The structure of a text fusen segment and a figure-drawing segment.

- selection (same as with a virtual object);
- copy or movement (same as with a virtual object);
- modification of the name (same as modifying a real object name);
- opening or setting parameters (same as opening a virtual object as a panel; the corresponding application is executed, and normally a dialogue panel is displayed); and

• modification of display attributes (same as modification of display attributes of a virtual object, except that the setting items are virtual object subsets).

As an example, a spreadsheet or database application adopting a tabular organization consists of cells. The attributes of each cell and information on table structures are stored as adjective fusen in TAD. The text data can be handled by any application, whereas only certain applications can process the adjective fusen.

TAD guarantees data exchange, at least of the data in each cell of the table-based application, as character strings. In this way various applications can make use of the table data at the most basic level, that of character-string data.

Variable-length segments. TAD consists of structured data called variable-length segments or variable-length data having the structure 0xFF + segment ID + data length + parameters, as shown in Figure 16. The 0xFF code at the beginning of the segment equates with the escape code in the TRON code system. 0xFF indicates the start of the segment, and the segment ID indicates the type of fusen. The data length gives the data size of the parameters that follow. The fusen segments and figure fusen segments contain a sub ID giving further information on the type of segment. In this case the structure appears as shown in Figure 17.

We gain two advantages from classifying types of fusen and adopting a structure that includes segment ID and byte-length fields. Applications unable to process a segment can skip it, based on data-length information, and go on to process the next data. This means that applications are not required to support every segment, and there is less burden on the system developers. Secondly, there is no need to maintain a large fixed-length table of formatting information.

Holding application-dependent data. The application-dependent data that are not covered by the TAD specifications can be stored in three possible ways: in the application program record, in the adjective fusen record, and in the text-application fusen or figure-application fusen.

The basic mechanism adopted in BTRON to associate the application-specific data and the application program itself is to embed a segment that holds the application ID. This ID designates the application that can interpret the data. The choice of the method to use in embedding the application ID is not specified at this moment. We believe the necessary underlying mechanisms support the idea of storing the application-specific data in TAD to promote the maximum portability.

Virtual object segments. TAD realizes the real/virtual object model by means of link records and virtual object segments. A link record holds link information to a real object, while a virtual object segment holds information for visually displaying a virtual object.

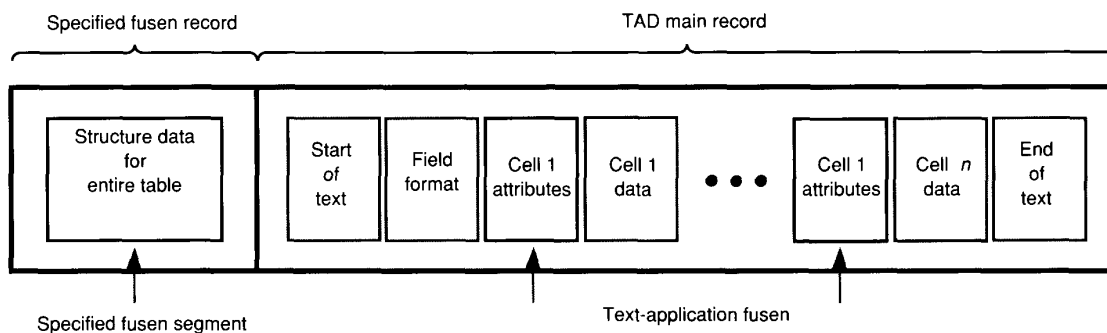


Figure 18. Data representation based on the real/virtual object model.

When data contain a virtual object, TAD saves the virtual object segment by embedding it in the text or figure data. The position of the segment shows the relative position of the virtual object in the data, and the order of appearance of real images defines the correspondence between link records and virtual object segments. Figure 18 shows how data are represented when two virtual objects exist in the same set of text data.

Other standard formats allow data in other files to be referenced, but TAD offers the advantages of referring to the network-type file and of preserving virtual information.

BTRON/286 implements the BTRON human-machine interface on an 80286-based hardware system. Please note that the BTRON human-machine interface can be implemented on different hardware platforms and can be implemented in many ways. Currently, work is proceeding to develop a BTRON human-machine interface that runs on the TRON VLSI CPU. In this implementation, designers emphasize distributed operating system functionality. We hope we can report on the next development in a year or two. 齋藤

References

1. K. Sakamura, "BTRON, The Business-Oriented Operating System," *IEEE Micro*, Vol. 7, No. 2, Apr. 1987, pp. 53-65.
2. K. Sakamura, ed., "TRON Project 1988: Open Architecture Computer Systems," *Proc. Fifth TRON Project Symp.*, Springer-Verlag, Tokyo, 1988.
3. *BTRON/286 Specification*, The TRON Association, Tokyo, Dec. 1988.



Yoshiaki Kushiki



Kazuhiro Oda

Ken Sakamura's biography, picture, and address appear on p. 13.

Yoshiaki Kushiki currently works in Matsushita Electric Industrial Co., Ltd.'s Information Systems Research Laboratory. He has been engaged in the research and development of system architecture, such as operating systems, databases, the human-machine interface, and artificial intelligence. He received the BE degree in electronics engineering from Kyoto University in Kyoto and is a member of the IEEE.

Kazuhiro Oda is a chief specialist in the Personal Computer Design Department at the Ome Works of Toshiba Corporation. He is now a member of the BTRON and CTRON technical committees of the TRON Association. He has been engaged in software design and development of large and medium-scale computers, small business computers, distributed processors, and word processors. Oda received the BE degree from Kyushu University in Fukuoka.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 153 **Medium** 154 **High** 155