US 20040103415A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2004/0103415 A1

Zuppicich (43) Pub. Date: May 27, 2004

(54) METHOD OF INTERFACING WITH DATA STORAGE CARD

(76) Inventor: **Alan N. Zuppicich**, Auckland (NZ)

Correspondence Address:
Patent Department
Choate, Hall & Stewart
Exchange Place
53 State Street
Boston, MA 02109 (US)

(21) Appl. No.: **10/715,979**

(22) Filed: **Nov. 18, 2003**

Related U.S. Application Data

(63) Continuation of application No. 09/676,591, filed on Sep. 29, 2000, now Pat. No. 6,698,654.

(30) **Foreign Application Priority Data**

May 9, 1995 (NZ)..... 272094

May 9, 1996 (WO)..... PCT/NZ96/00038

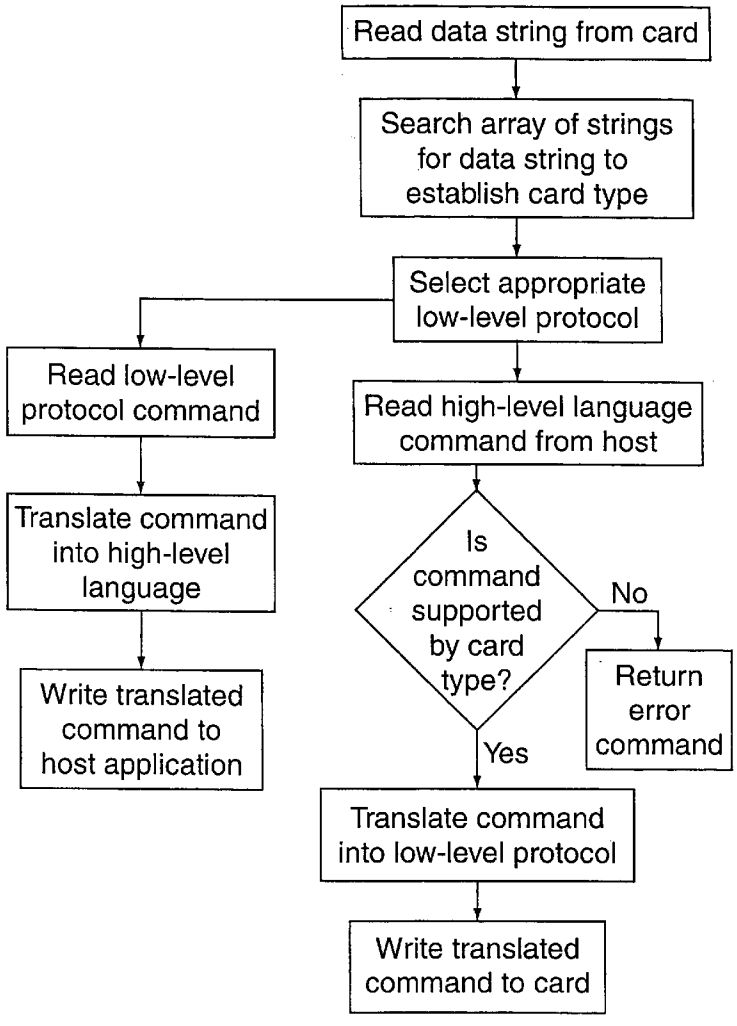
Publication Classification

(51) **Int. Cl.**[7] **G06F 9/00**

(52) **U.S. Cl.** **719/310**

(57) **ABSTRACT**

A card reader/writer which interfaces between a host application program and a data storage card. The card reader/writer reads from and writes to the application program with high level language of the commands. The card reader/writer translates the high level language commands of the host program to corresponding sequences of low level commands for reading and writing to the data storage card. The card reader/writer stores a plarality of sets of such low level commands, and is able to establish the card type for any card interfaced with the card reader/writer, and to use the appropriate command set for the established card type. The card reader/writer also translates low level commands from the card to high level commands for the host application program. The card reader/writer can be loaded with low level command sets for additional card types as required.



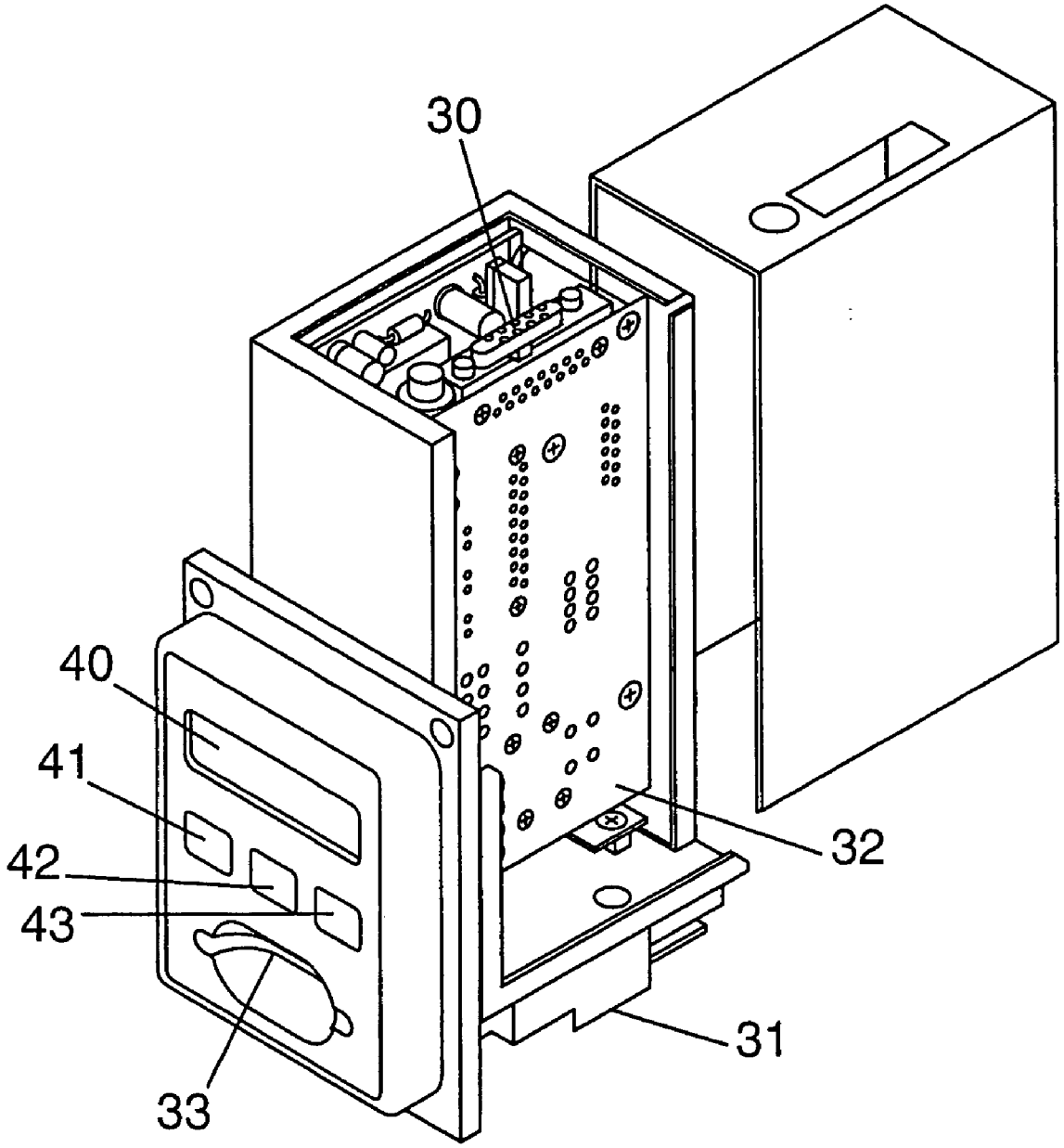


FIG. 1

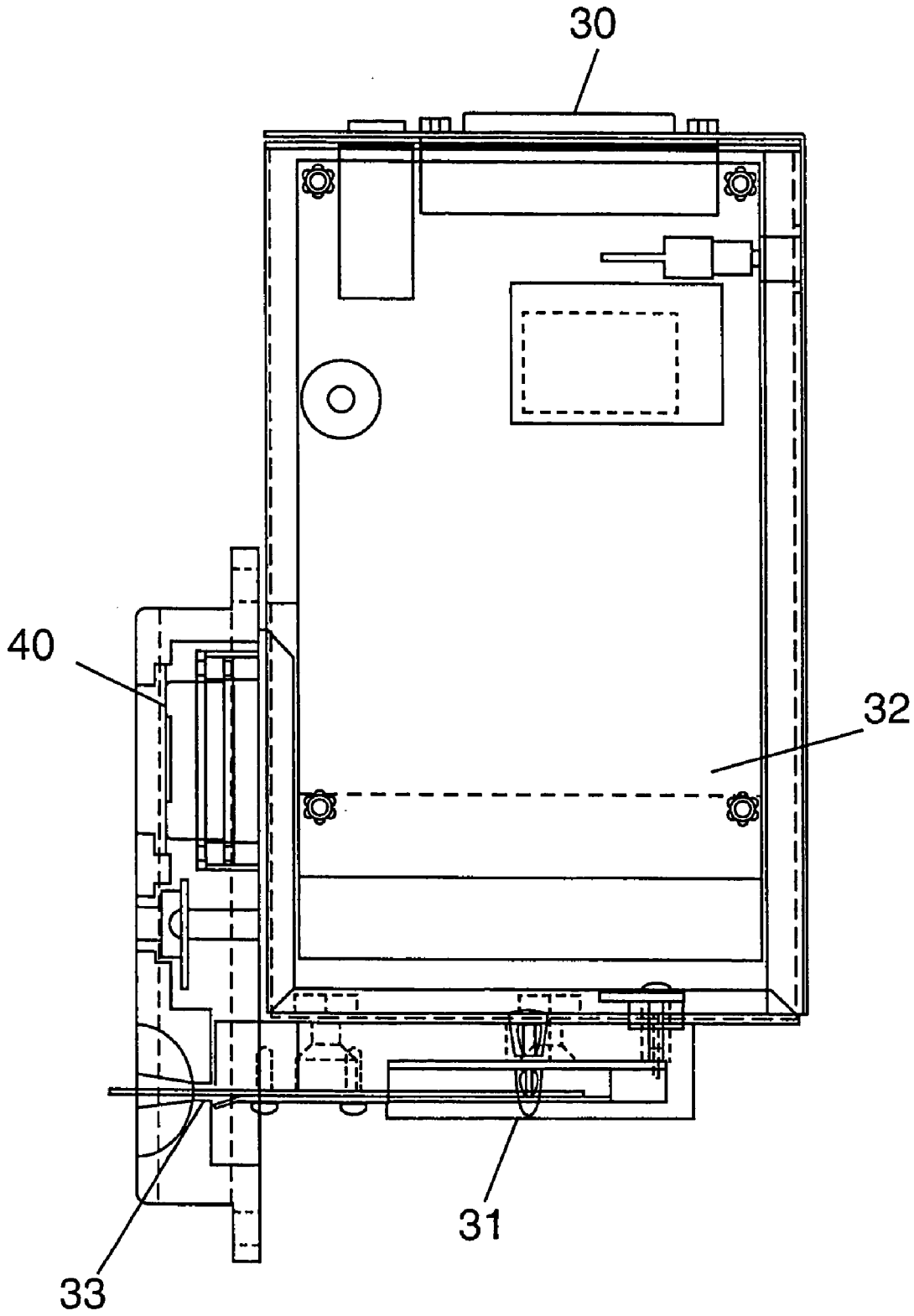


FIG. 2

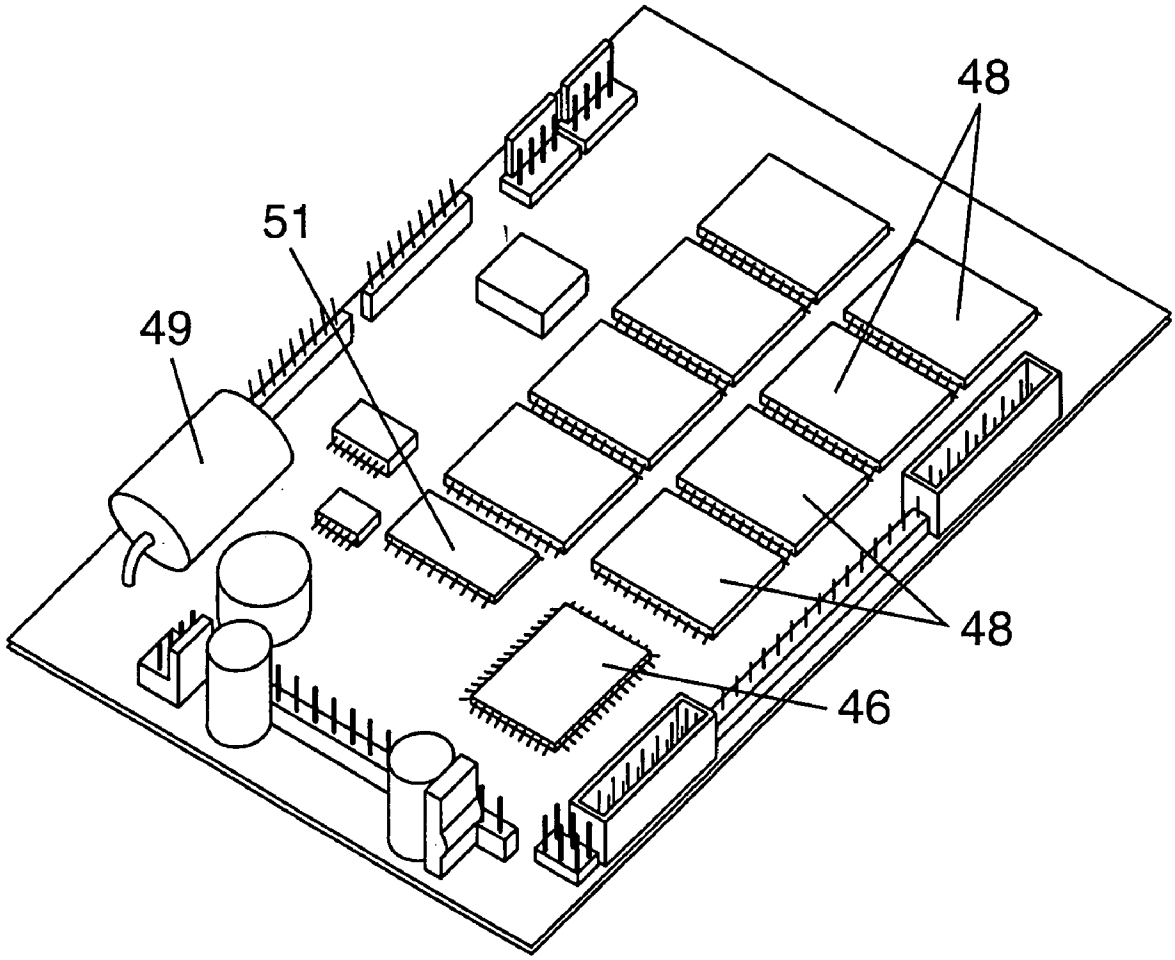


FIG. 3

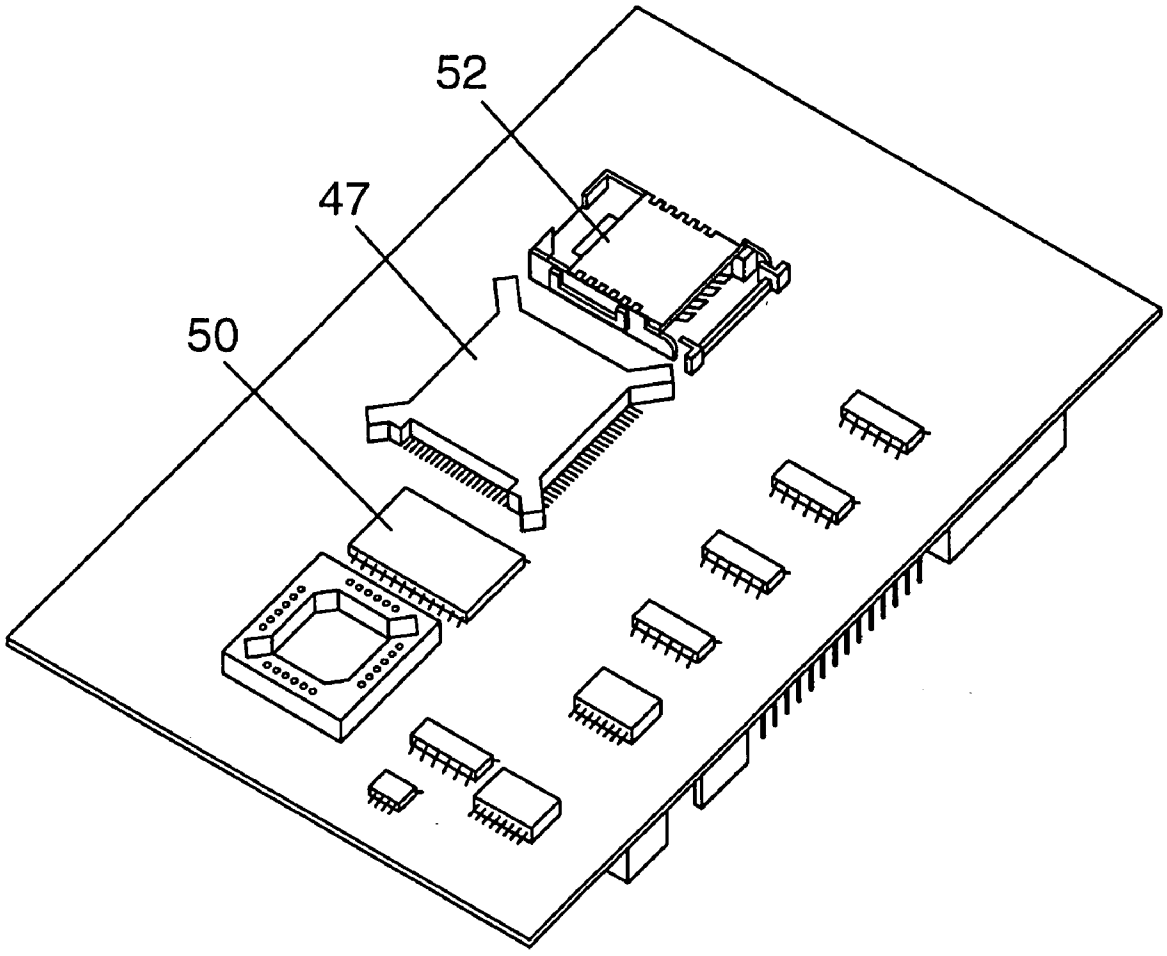


FIG. 4

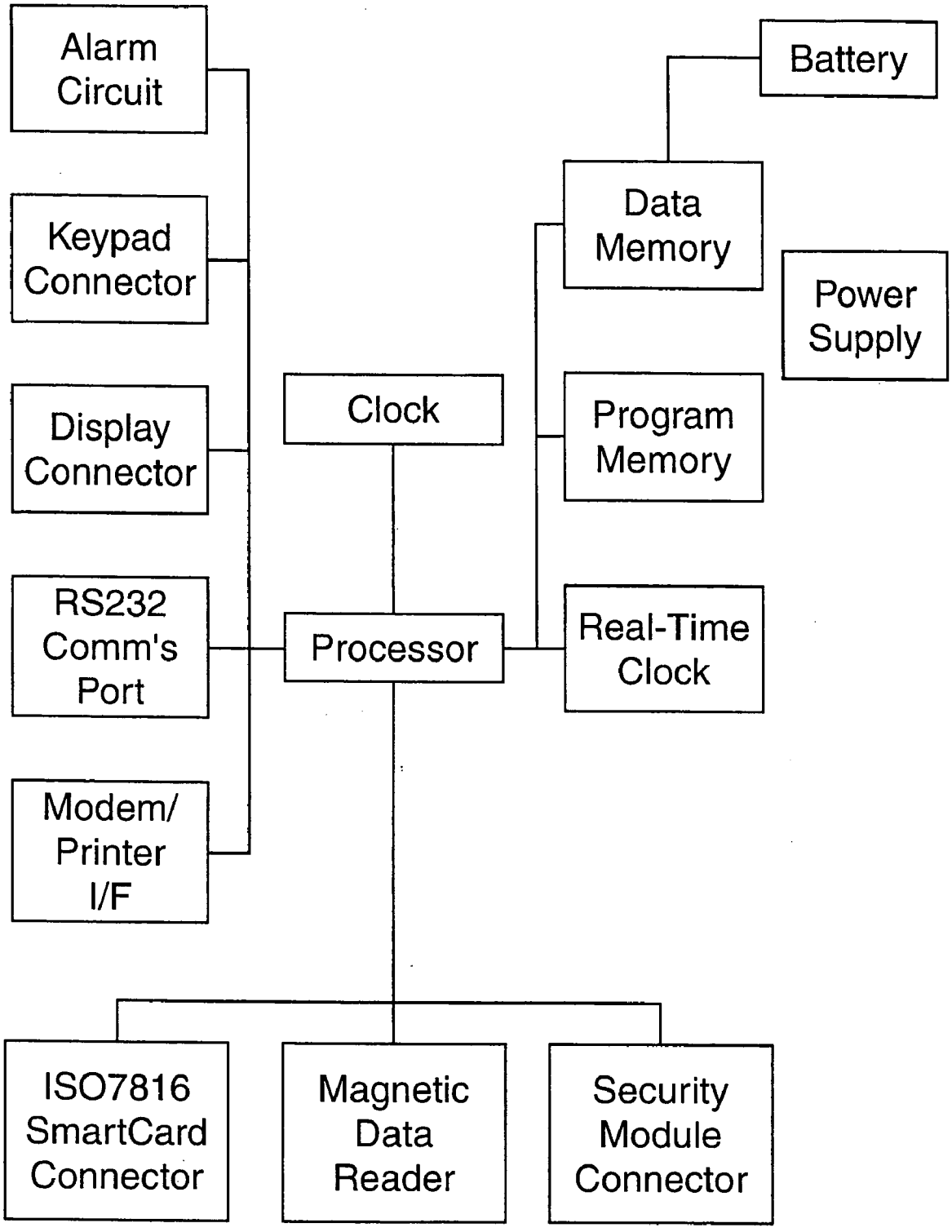

Alarm Circuit
Keypad Connector
Display Connector
RS232 Comm's Port
Modem/ Printer I/F
Clock
Processor
Battery
Data Memory
Power Supply
Program Memory
Real-Time Clock
ISO7816 SmartCard Connector
Magnetic Data Reader
Security Module Connector

FIG. 5

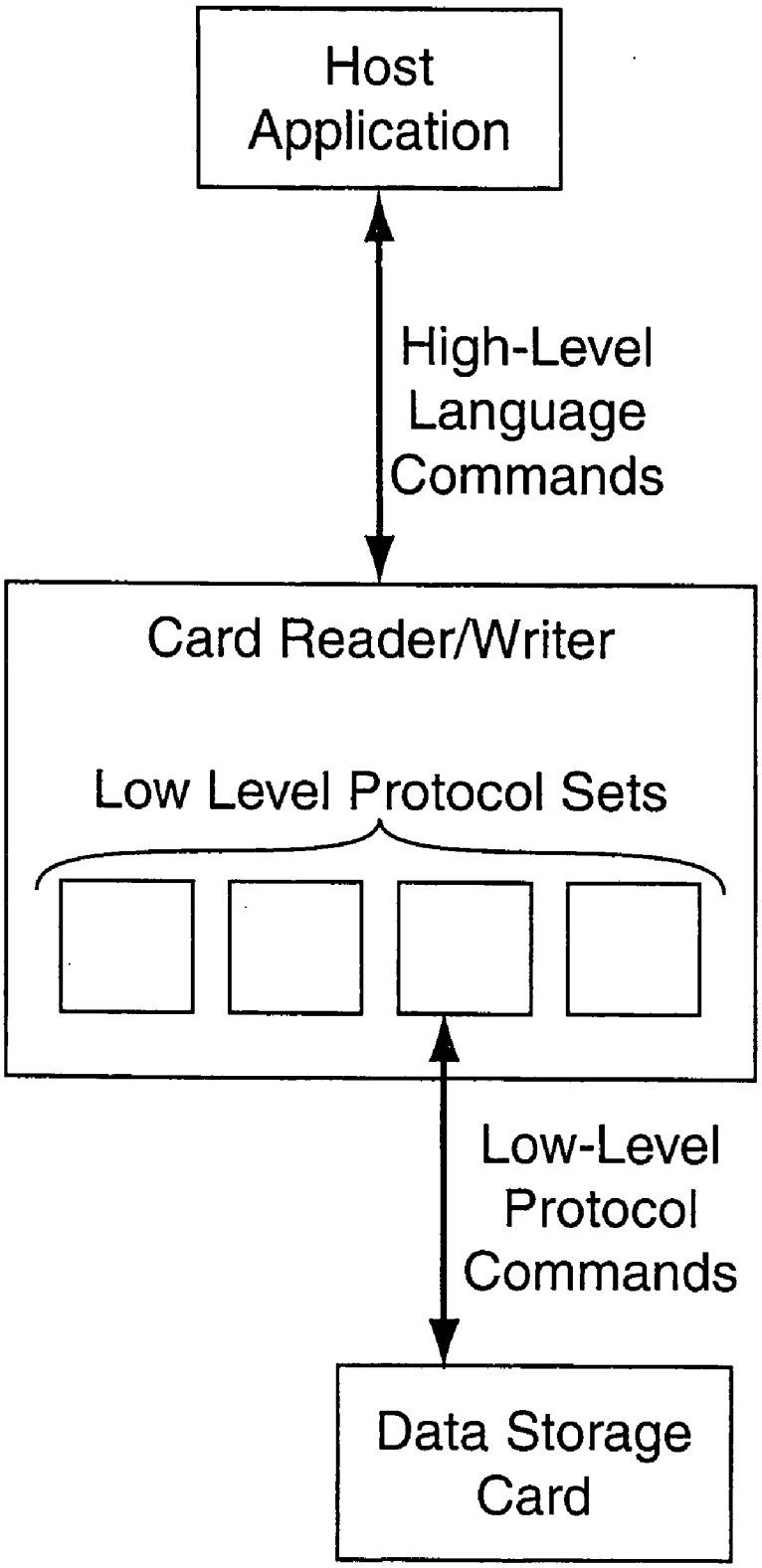

Host Application
High-Level Language Commands
Card Reader/Writer
Low Level Protocol Sets
Low-Level Protocol Commands
Data Storage Card

FIG. 6

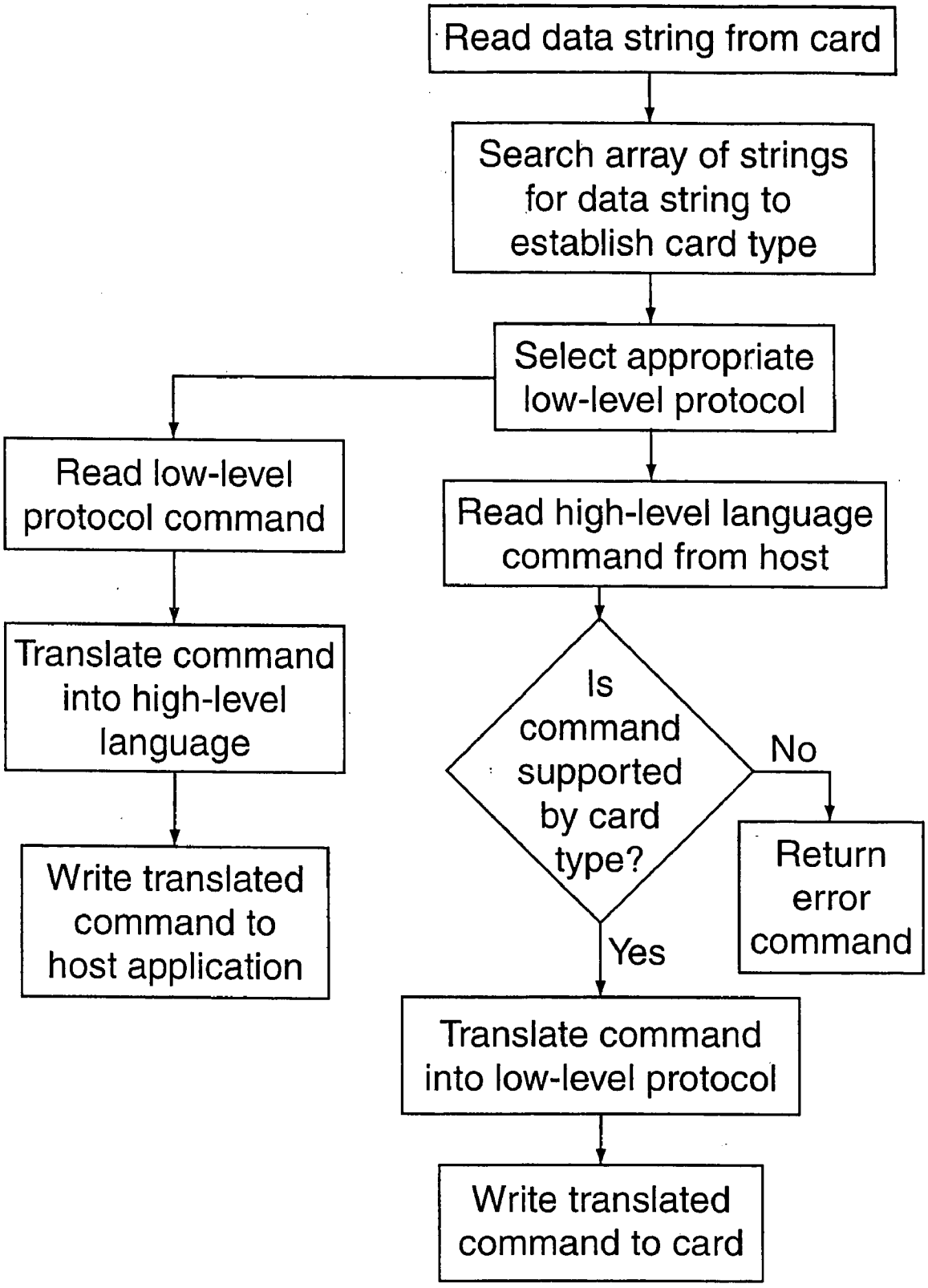

Read data string from card
Search array of strings for data string to establish card type
Select appropriate low-level protocol
Read low-level protocol command
Translate command into high-level language
Write translated command to host application
Read high-level language command from host
Is command supported by card type?
No
Return error command
Yes
Translate command into low-level protocol
Write translated command to card

FIG. 7

First Array

Data String	Card Identifier
X X X ⋮	X X X ⋮

Second Array

Card Identifier	Commands
X X X	1. X X X 2. Y Y Y 3. Z Z Z
Y Y Y ⋮	1. X X X ⋮

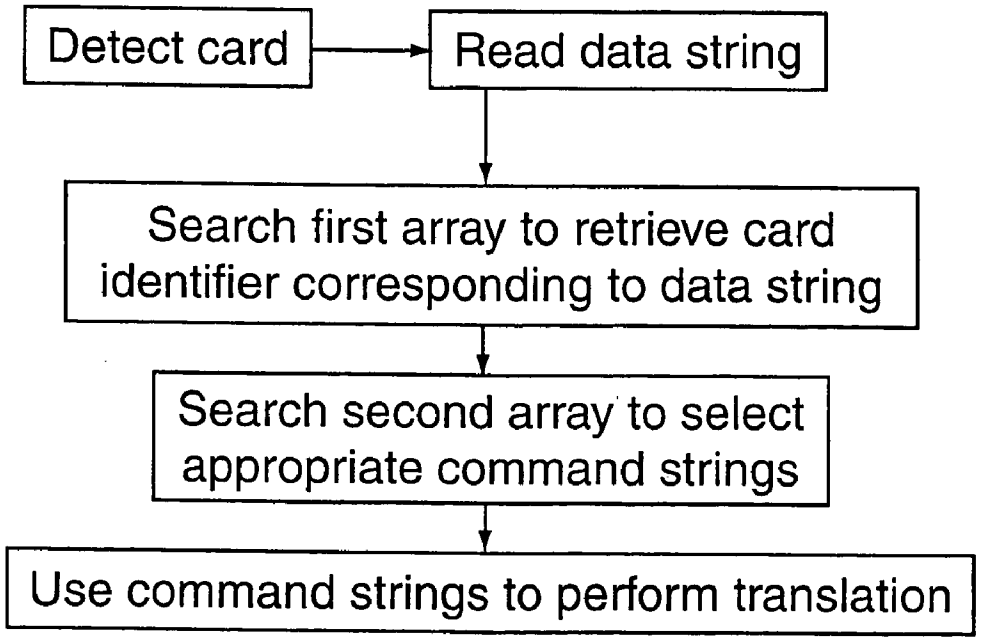


FIG. 8

METHOD OF INTERFACING WITH DATA STORAGE CARD

TECHNICAL FIELD

[0001] This invention relates to cards having magnetic or electronic data storage ("Smartcards") capability and in particular to interfaces between such cards and an application specific programmed controller.

BACKGROUND ART

[0002] Smartcards and smartcard readers are disclosed in patents to Innovatron and Bull CP. See for example U.S. Pat. Nos. 4,102,493 and 4,404,464. Smartcards generally conform to one or more parts of ISO standard 7816.

[0003] Several prior art smartcard reader/writers use a programmed microprocessor to perform a series of predetermined actions on a smartcard under control of an external control system. The external control system must have "full knowledge" of the smartcard being used. Any changes to the cards being used must be reflected in changes to the external control system. The designer or user of the external control system must have an extensive understanding of the smartcards for which the reader/writer is intended.

[0004] Several other prior art smartcard reader/writers incorporate a programmed microprocessor to perform all actions on a smartcard autonomously. These reader/writers typically form a completed product, i.e. vending machine or point-of-sale terminal. Alteration of the product to support new or different smartcards requires a redesign or modification of the fundamental reader/writer component. This restricts the ability of an original equipment manufacturer to incorporate smartcard technology in traditional products.

[0005] Existing smartcard reader/writers fall into two categories. The first is that which acts as card coupler. A card coupler only provides the hardware and minimum necessary software to read from, or write to, a smartcard. The coupler may support a range of cards, possibly from more than one card manufacturer. However, this support is, because the final use is unknown, largely confined to the simplest and most generalised tasks. The coupler is commonly used as a development tool for entry into the smartcard field, or as a means of adding smartcard capability to existing product designs. In either case the user of the coupler must gain or possess a good knowledge of the smartcard, the card data structures, and data transfer procedures.

[0006] The other extreme is that of an application or product with imbedded smartcard capability. The reader is an intrinsic part of the host control system. Support is limited to a few card types, with data contents, structures, security and transfer procedures predetermined by the equipment manufacturer.

[0007] There is currently no easy way for smartcard capability to be added to a manufacturers product or for azn existing card reader to be altered to handle new smart card types, short of that manufacturer becoming an expert in the smartcard field. Should a competitor proceed down this path, the delay in recovering to a competitive position would be costly and time consuming. Furthermore there is currently no card reader available which can read a variety of types of magnetic cards as well as a variety of types of chip cards.

DISCLOSURE OF INVENTION

[0008] It is therefore an object of the present invention to provide an interface for use between cards having magnetic or electronic data storage capability and an application specific program controller that at least goes some way toward overcoming the above disadvantages.

[0009] In a first aspect the invention consists in a card reader/writer which interfaces a host application program with a data storage card characterised in that:

[0010] said card reader/writer can respond to said application program using at least one designated high level language,

[0011] a plurality of low level protocol sets are stored which each correspond to a known data storage card type,

[0012] said card reader is able to establish the card type for any card interfaced to it for which it has a protocol set, and select from its store of protocols the appropriate low level protocol for the established card type,

[0013] said card reader/writer reads and translates high level language commands from the host program to corresponding commands within said established low level protocol and writes these low level commands to said card,

[0014] and said card reader/writer reads commands in said established low level protocol from said card translates them to a corresponding command in said high level language and writes these commands to said host application program.

[0015] In a second aspect the invention consists in a method of interfacing a host application program with a data storage card comprising the steps of:

[0016] establishing the card type,

[0017] selecting from a store of protocols the appropriate low level protocol for the established card type,

[0018] reading high level language commands from the host program,

[0019] translating said read high level language commands to corresponding commands within said established low level protocol,

[0020] writing said corresponding commands to said card,

[0021] reading commands in said established low level protocol from said card,

[0022] translating said low level protocol commands into corresponding commands from said high level language, and

[0023] writing said corresponding commands from said high level language to said host application program.

[0024] In a third aspect the invention consists in a method of enabling a controller to read and write data via a card read/write station to a plurality of known types of card devices incorporating magnetic or electronic data storage

means, which method uses a stored-program stored-data processor, said method comprising: storing first data arrays containing strings of known card characterising data and for each string its corresponding card identifier, storing second data arrays containing card identifiers and for each identifier its corresponding command strings, and executing a program on said processor which causes said processor to:

[0025] (1) detect the presence of a card in said read/write station and to pass a card detect signal to said controller,

[0026] (2) read the data string which characterises the card,

[0027] (3) search through at least one said first array for a matching data string, and upon making a successful match to retrieve the corresponding card identifier,

[0028] (4) select the appropriate command strings in said second array using the retrieved card identifier,

[0029] (5) accept generic transaction instructions from said controller,

[0030] (6) translate the instructions from said controller to commands appropriate to the inserted card using the selected command strings from the second array, and

[0031] (7) either read or write data to said card in accordance with said generic instructions.

[0032] In a fourth aspect the invention consists in a universal card interface for interfacing smartcards and other card devices having electronic or magnetic data storage with a controller requiring read/write access to said cards comprising:

[0033] a card read/write station into which cards may be inserted for data transfer, a processor having associated memory and input-output ports,

[0034] said read/write station connected to one input-output port and said controller connected to a second input-output port,

[0035] interface software and data arrays stored in said memory,

[0036] said data arrays including first and second data arrays with said first data arrays containing strings of known card characterising data and for each string its corresponding card identifier, said second data arrays containing card identifiers and for each identifier its corresponding command strings, said software when executed by said processor causing said processor to:

[0037] (1) detect the presence of a card in said read/write station and to pass a card detect signal to said controller,

[0038] (2) read the data string which characterises the card,

[0039] (3) search through at least one said first array for a matching data string and upon making a successful match to retrieve the corresponding card identifier,

[0040] (4) select the appropriate command strings in said second array using the retrieved card identifier,

[0041] (5) accept generic transaction instructions from said controller,

[0042] (6) translate the instructions from said controller to commands appropriate to the inserted card using the selected command strings from the second array, and

[0043] (7) either read or write data to said card in accordance with said generic instructions.

[0044] In a fifth aspect the invention consists in a method of enabling application software to interface with any known type of smartcard or other card devices having electronic or magnetic data storage, comprising:

[0045] (1) reading from the card to be interfaced the data string which characterises that card,

[0046] (2) searching through a first data array containing strings of known card characterising data, and for each string its corresponding card identifier, for a matching data string, and upon making a successful match retrieving the corresponding card identifier,

[0047] (3) selecting the appropriate command strings from a second data array containing card identifiers, and for each identifier its corresponding command strings, using the retrieved card identifier,

[0048] (4) accepting generic transaction instructions from said application software,

[0049] (5) translating the instructions from said application software to low level commands appropriate to the card being interfaced using the selected command strings from said second data array, and

[0050] (6) causing data to be either read from or written to said card in accordance with said generic instructions.

[0051] In a sixth aspect the invention consists in a software algorithm which enables application software to interface with any known type of smartcard or other card devices having electronic or magnetic data storage, comprising a hierarchy of functional modules wherein:

[0052] a first level module reads from the card to be interfaced the data string which characterises that card and passes said data string to a second level module, the second level module searches through a first data array containing strings of known card characterising data, and for each string its corresponding card identifier for a matching data string, and upon making a successful match retrieves the corresponding card identifier and passes said card identifier to a third level module,

[0053] the third level module selects the appropriate command strings from a second data array containing card identifiers, and for each identifier its corresponding command strings, using the card identifier passed to it,

[0054] a fifth level module accepts generic transaction instructions from said application software and passes those instructions to a fourth level module,

[0055] the fourth level module translates the instructions from said fourth level module to low level commands appropriate to the card being interfaced using the command strings selected by the third level module, and

[0056] the first level module either reads from or writes to said card in accordance with the translated generic instructions passed from the fourth level module.

[0057] The universal card interface of the present invention is able to recognise any ISO card type presented to it, communicate with the card using the protocols appropriate to that-card type, and communicate with the host application program using a high level command language.

BRIEF DESCRIPTION OF DRAWINGS

[0058] A particular embodiment of the present invention will now be described with reference to the accompanying drawings in which

[0059] **FIG. 1** is a perspective view of the apparatus of the preferred embodiment of the present invention,

[0060] **FIG. 2** is a side elevation in cross-section of the apparatus of the embodiment of **FIG. 1**,

[0061] **FIG. 3** is a perspective view of the printed circuit board (PCB) of the control system assembly of the preferred embodiment of the present invention,

[0062] **FIG. 4** is another perspective view of the PCB of the preferred embodiment of the present invention, and

[0063] **FIG. 5** is a block diagram of the electrical and electronic apparatus of one embodiment of the present invention.

BEST MODES FOR CARRYING OUT THE INVENTION

[0064] One embodiment of the present invention is a Universal Card Interface (or UCI) being a software-controlled card interface device that supports a wide range of smartcards and credit cards.

[0065] Referring to FIGS. **1** to **5**, the UCI has two major elements, a physical card accepting device **31** and a control system assembly **32** which includes all of the electronic circuitry required for controlling card accepting device **31**. Card accepting device **31** has a single slot **33** which accepts all card types. The form taken by the card accepting device will depend largely on the application intended for the UCI. For example, the size of display and number configuration of buttons on the keypad, configuration of the card slot **33** and mode of physical acceptance of the card (whether the card is fully retracted into the card accepting device or is allowed to remain partially outside the device) are all matters capable of variation. Control system assembly **32** is intended to be independent of application and is capable of supporting card accepting devices of the abovementioned variations.

[0066] Referring to **FIGS. 1 and 2**, the card accepting device of the particular embodiment of the present invention has been designed to be easily fitted to existing vending machines, and in particular has been designed to replace standard bank note validator spaces on such machines. The card accepting device has a friendly user interface including a 2 line LCD display **40** which may be in either green or yellow for improved visibility in dark conditions, a user input panel with 3 push buttons (OK **43**, BALANCE **41**, and CANCEL **42**), which preferably provide a tactile response to the user, and a card insertion slot **33**. The card insertion slot **33** has a push-pull acceptor, a clearing space for foreign bodies, landing contacts, sealed detector switch, and accepts cards by horizontal insertion, with the chip side up.

[0067] Referring to FIGS. **3** to **5**, the UCI control system assembly incorporates a primary microprocessor **46** and a support microprocessor **47**. The primary microprocessor **46** may for example be a Dallas DS5002 FP-16. The support microprocessor **47** may for example be a Motorola MC68302.

[0068] The example primary processor is derived from industry standard "8051" architecture, with a secure memory protection mechanism to combat against accidental or deliberate tampering or viewing. As with the 8051, the memory is divided into that used by the program and that used for storage of data. This memory **48** is all battery-backed static RAM, the battery **49** being a long-life lithium cell. The memory **48** is automatically switched to battery and write-protected as the power fails. The processor is protected against erroneous actions by a watch-dog timer. The memory capacity for both data and program are separately configurable. Options range in each case as necessary from 128 Kbytes to 512 Kbytes.

[0069] The example support processor is based on the industry standard "68000" architecture, with enhancements for high speed data communication. No sensitive data is held in this memory, with this processor being responsible only for low security functions such as magnetic card reading and data transfer. The processor is protected against erroneous actions by a watch-dog timer. Memory 50 consists of 128 Kbytes of static RAM for program and data. Battery backup of this memory 50 is not necessary, as it is reloaded from the primary microprocessor as required. A 32 Kbyte ROM contains the embedded bootloader. All software except the embedded bootloader is remotely reloadable.

[0070] The UCI electrical subassembly further includes a real time clock **51** such as a Dallas DS1293S. The real time clock provides accurate time of day and calendar functions. It operates from an independent clock at 32.768 kHz and receives power from the primary microprocessor **46**.

[0071] A small socket **52** on the PCB allows a SAM (a physically cut down smartcard) to be added to the board as required. This SAM can be used to hold encryption keys or a cypher algorithm in high security applications. The primary microprocessor **46** may also perform this task, but commercial issues may make using a SAM preferable.

[0072] The UCI PCB has connection adapted to receive a 1 or 2 track magnetic stripe reader. The UCI provides 5 volt power to the reader and accepts standard TTL logic level signals.

[0073] The reader will accept cards confirming to ISO 7811 or ISO 7813. The UCI can read two tracks simultaneously, these being track **2** and either track **1** or track **3**.

[0074] The smartcard acceptor circuit is designed to support cards conforming to ISO 7816 with the addition of a suitable mechanical card acceptor mechanism. With the correct card acceptor, hybrid "magnetic and chip" cards can be accepted through a common slot. This is desirable in markets where gradual migration is required. Both synchronous (memory, or token) cards and asynchronous (microprocessor) cards can be accepted. The programming voltage (VPP) is preferably limited to 5 volts which is expected by modem cards.

[0075] Other user interface options which are preferably supported by the UCI include:

[0076] Keypads connectable directly to the UCI. Generally only matrix keypads will be required, however supporting larger keypad configurations would require little extra effort.

[0077] Liquid crystal display (LCD) connectable directly to the UCI. Most LCDs utilising the Hitachi HD44780 chip (an industry standard) or its equivalents are suitable. These displays are available in various formats from 1 line, 16 character, up to 4 line, 24 character.

[0078] A status indicator. Preferably two light emitting diodes (LED) are positioned at the front of the UCI to provide a low cost indication of the UCI's status. The LEDs if used would protrude through the card slot facia at either end of the card slot, doubling as status indicator and highlighting the card slot position.

[0079] An audible feedback indicator. A buzzer is contained on board to provide audible feedback for keystrokes or error conditions. An example of suitable buzzer characteristics would be a buzzer generating a sound pressure level of 90 dB (@0.1 m) at a nominal frequency of 3100 Hz.

[0080] The UCI is a uniform interface to third party equipment in which it is to be included. The vendor interface includes three distinct elements, a high voltage relay, a current loop serial bus and an RS232 serial interface. Furthermore, the card accepting device supports serial data modem connection, or printer connection, via RS485/RS232 connector **16**.

[0081] The control system assembly includes two power sources, in particular a on-board battery which provides backup power to the primary microprocessor memory and to the clock along with a regulated mains power supply. A suitable battery for the battery power supply might for example be a 3.6 volt 1200 mAHr battery.

[0082] The UCI of the present invention has the ability to interface with a plethora of cards, both smart and magnetic, to understand these cards, and to perform whatever action is required, with the absolute minimum of instruction. It can read ISO 7811 credit cards track **1** or track **2**, read and write to ISO 7816-3 (T=0 and T=1) smartcards, and read and write ISO 7816 memory cards.

[0083] In traditional smartcard couplers, the electronic support of the card was limited to very simple tasks, because the end use of the coupler was unknown. The present invention, however, extends the abilities of the coupler concept with added layers of software (the microprocessor program), each layer building in more functionality, and at the same time, simplifying the task of communicating with the card. To add these layers and perform a useful function with them, the UCI includes a format database, which contains information on each type of card that the UCI may accept. This information forms the rules and guidelines for use of that card. It includes a summary of the card data structures, transfer procedures, data contents, and security information. The guidelines may refer the UCI to a security module if needs dictate. The security module is a small microprocessor device similar to a smartcard which is a semi-permanent part of the PCB.

[0084] With the information available from the format database, the UCI can easily determine the overall functionality of the UCI and card combination. This can modify the sequence of actions taken internally in response to any external instruction. The external device does not need to appreciate the differences between various types of cards. Because the external device does not issue wildly differing instructions with each new card presented, the software interface between UCI and the external device is simplified.

[0085] The software interface can be further simplified by recognising that a limited number of instructions will perform the great majority of desired actions on the cards. The top most software layer accumulates all the necessary instructions or commands to define the software interface. The information in the format database is also used to check for any potential breach of system security when using instructions at a lower software layer.

[0086] The software is organised in layered, modular structures as shown in the following table.

TABLE 1

Layers of the Microprocessor Program					
		Access to			
Layer	Purpose	Encryption, Security	Card Format Library	Card Primitive Library	Applicable Standards
8 Application	Interface to external environment				
7 Presentation	Uniform interface to Application layer		Y		

TABLE 1-continued

Layers of the Microprocessor Program					
		Access to			
Layer	Purpose	Encryption, Security	Card Format Library	Card Primitive Library	Applicable Standards
6 Process Procedures	General interface to all card activity		Y		AS 2805
5 Process Primitives	Support routines	Y	Y		
4 Entity Procedures	Interface to all card data structures, i.e. Files, Purses, Directories	Y	Y		
3 Entity Primitives	Support routines for Files, Purses, Directories		Y	Y	
2 Stream Control, ISO level Commands	Interface at card (ISO) command level			Y	ISO 7816 Part 3, ISO 7816 Part 4
1 Low Level Primitives, Hardware Drivers	Hardware support, and control. Card interface at bit/byte level			Y	ISO 7810, ISO 7816 Part 1, ISO 7816 Part 2

[0087] As can be seen from the above table in the preferred embodiment, the operations of the UCI are split into eight layers.

[0088] Each layer can be considered as consisting of collections of objects, where an object is a self-contained arrangement of functions and data. The objects in each layer are generally only accessible to the layer immediately above or below. The interface between layers is well regimented with controls based on the format database information preventing ill-advised operations.

[0089] The data contained in the format database is only available to selected objects and layers. This minimises the possibility of the external device gaining sensitive knowledge of particular card formats. Similarly, access to the security and ciphering program modules is also restricted to selected objects and layers.

[0090] The format database consists of two primary areas, the format library, and the primitive library. The primitive library contains pre-personalisation details of all cards accepted. This includes card capacity, electrical signal timing, character interchange protocols, and a list of the features available on the basic card. This information is that supplied by the card manufacturer. The format library contains the details added to a card by the issuer, after the card has left the manufacturer.

[0091] Having regard now to the operations of each layer, the layers range from the primarily low level character oriented operations of layer 1 to the high level generic interface of layer 8.

[0092] Layer 1 responsibilities include the basic actions required for card insertion or removal and the simple transmission of data to and from the card.

[0093] Insertion of a card activates a micro switch which is constantly monitored. Card insertion is signalled to higher layers, so that appropriate action can be taken. Higher layers pass down the instruction to attempt to identify the current card. The card then has power and control signals applied according to ISO 7816. When the card is no longer required, the card interface circuitry is powered down in a sequence specified by ISO 7816. If the card is removed prematurely, the same sequence is followed as quickly as possible. Higher layers are informed of the absence of the card, so that further "closing" actions may be taken. This may include deletion of session keys, exception logging, or alerting the cardholder. Another function that is incorporated at this level is encryption or decryption of the data stream.

[0094] Transfer of individual characters and bits of data to and from the card is done at this layer, but grouping of characters into messages and commands is the domain of higher layers. In general, UCI hardware is dealt with at this level only on a character by character basis. Coping with the intelligence contained within data streams, messages, commands and the like is left to the higher code layers.

[0095] Layer 2 of the UCI controls the interaction with the card at a card command level. Level **2** for example is adapted to correctly format commands and data for transmission to the card by layer 1. Message formatting, error detection, and transmission characteristics for the smartcard are performed by layer 2 to the specifications of ISO 7816.

[0096] Support for the various UCI features such as time of day clock, buzzer, communications ports, display, and keypad are enhanced by code in this layer. Other support code found here includes printer drivers, display drivers, modem drivers, keypad buffer, and ISO 7810 magnetic card reader data buffer.

[0097] During the card insertion phase, layer 2 is used to initiate the reset process for the card. The card's response to the reset process is used to determine whether the card communicates in a synchronous or asynchronous manner. This information is used by layer 3.

[0098] Interfacing to layer 3 is through "traps" or software interrupts. Layer 1+2 traps exist for: real-time clock, system, keypad, LCD, communication ports, magnetic card reader, smartcard/security module.

[0099] The operation of layer 3 of the UCI is dependent on the card type. Therefore it is important to correctly identify the card type, dependent on whether it is a synchronous card (typically token cards, or memory cards, non-microprocessor based) or a microprocessor based asynchronous.

[0100] In identifying a synchronous card, the first three bytes are read from the card. These are compared with 3, 2 or even 1 byte strings contained in one column of a small array. In an adjacent column are reference numbers to card offerings from various card manufacturers. A match to any string indicates that the card is known, and allowed. Table 2 shows an example of such an array. Having now identified that the card is a synchronous card of a particular "model" or type, from a particular manufacturer, the UCI can select only those commands and handling procedures applicable to this card. The higher layers are therefore informed of the card type found.

TABLE 2

String to Match	Card Type
123	type 3
56E	type 1
AB	type 12
C	type 3
7	type 18

[0101] In identifying an asynchronous card, the Historical Bytes of the ISO 7816 Answer To Reset string are compared with strings contained in one column of a small array. In an adjacent column are reference numbers to card offerings from various card manufacturers. A match between strings indicates that the card is known, and allowed. An example of such an array is shown as Table 3.

TABLE 3

String to Match	Card Type
24, 10, 00	type 6
30, 11, 03	type 2
31, 1F, FF	type 15
4C, IF, FF	type 7
4D, 1F, FF	type 19
23, 00	type 7

[0102] Having now identified that the card is an asynchronous card of a particular "model" or type, from a particular manufacturer, the UCI can select only those commands and handling procedures applicable to this card. The higher layers are therefore informed of the card type found.

[0103] Access to the processes outlined above is made through the function:

[0104] int decide_card_type(void)

[0105] Returns: M_CARD_TYPE, or NULL if card type not known.

[0106] In communicating with a synchronous card, layer 3 uses a function "Access_Sync_Card" to control the communication. For example:

[0107] int Access_Sync_Card(int command, int card_type, int len, char *pointer)

[0108] where command is selected from READ_UPDATE

[0109] ERASE_UPDATE

[0110] READ_FIXED

[0111] READ_TOKEN

[0112] READ_MANUFACTURER

[0113] DECREMENT_TOKENS

[0114] PRESENT_SECRET_CODE

[0115] card_type is Card Type as determined in L3.1, L3.3,

[0116] len is the number of bytes to be taken from buffer, or the number of data bytes expected back from the card,

[0117] *pointer points to a buffer containing data for the card, or data from the card at completion of the function call, and the function returns

[0118] NO_ERROR if the command is successful or

[0119] ERR_CARD_LOCKED or

[0120] ERR_FUNCTION REFUSED or

[0121] ERR_THREE_BAD_PRESENTATIONS if the command is unsuccessful.

[0122] Based on the Card Type, and the Command, the function "Access_Sync_Card" refers to a table of "action string" pointers as shown in Table 4.

TABLE 4

A NULL pointer indicates that the Command is invalid for the current card.

	Command				
Card Type	READ_ UPDATE	READ_ FIXED	READ_ TOKEN	READ_ MANU-FAC-TURER	DECREMENT_ TOKENS
1	0	s1	s2	s4	s8
3	s5	s1	s2	s4	s8
4	s5	s1	s3	s4	s7
5	0	s12	0	s13	0
12	0	s11	0	s13	0
13	0	s15	0	s14	0
17					
18					

[0123] The pointers s1 etc of Table 4 refer to a further table, for example Table 5, this time of "action strings". An "action string" may be used by one or more cards.

TABLE 5

Pointer	Action String
s1	
s2	
s3	
s4	READ_128, RESET_ADDRESS, STEP 8, ERASE 32, WRITE 32, NULL
s5	RESET_ADDRESS, STEP 10, READ 32, NULL
s6	RESET_ADDRESS, STEP 10, WRITE 16, NULL
s7	STEP 96, NULL
s8	
s9	

[0124] An "action string" consists of a string of 1 or more "tags", with a NULL terminator. Most "tags" are followed by a "count" value. The "count" determines the number of card bits or clock cycles the "tag" should act upon. For example "action strings" are of the format:

[0125] [tag[,count],][tag[,count],] NULL

[0126] where tag is one of RESET_ADDRESS

[0127] STEP count

[0128] READ count

[0129] ERASE count

[0130] WRITE count.

[0131] In communicating with an asynchronous card, layer 3 uses a function "Access_Async_Card" to control the communication. For example:

[0132] int Access_Async_Card(int command, int card_type, int len, char *pointer, int P1, int P2)

[0133] where command may be selected from READ_FILE

[0134] WRITE_FILE

[0135] SELECT_FILE

[0136] SELECT_DIRECTORY

[0137] SELECT_KEY

[0138] UPDATE_FILE

[0139] PRESENT_SECRET_CODE

[0140] card_type is Card Type as determined in L3.2, L3.3,

[0141] len is the number of bytes to be taken from buffer, or the number of bytes expected back from the card,

[0142] *pointer points to a buffer containing data for the card, or data from the card at completion of the function call,

P1:	argument 1: for example the offset into file
P2:	argument 2: for example the selected file number

[0143] and the function returns one of:

4001	ERR_FUNCTION_INVALID
4002	ERR_FUNCTION_MISMATCH
4xxx hex	other error conditions detected by UCI code
6xxx hex	= ISO and Manufacturer error codes (no translation)
9000	NO_ERROR
9xxx hex	= ISO and Manufacturer error codes (no translation),

[0144] the function returning "NO_ERROR" if the command is executed successfully.

[0145] Based on the Card Type, and the Command, the function "Access_Async_Card" refers to a table of "format string" pointers as shown in Table 6.

TABLE 6

A NULL pointer indicates that the Command is invalid for the current card.

	Format String for Command				
Card Type	READ_FILE	WRITE_ FILE	SELECT_ FILE	SELECT_ DIRECTORY	SELECT_KEY
6	0	f21	f12	f14	f18
COS16DES	f15	f21	f12	f14	f18
PCOS	f15	f21	f13	f14	f17
2	0	f32	0	f33	0
MCOS	0	f31	0	f33	0
COS	0	f35	0	f34	0
15		0	f39	0	f41
19					

[0146] The pointers refer to a further table such as Table 7 of "format strings". A "format string" may be used by more than one card.

TABLE 7

Pointer	Action String
f154	51 80 3A 00 P2 00
f12	55 80 3A 00 03 00
f14	63 80 34 00 00 08

TABLE 7-continued

Pointer	Action String
f18	41 84 DA 00 P2 LEN
f31	
f33	
f35	
f39	
f41	

[0147] The format strings are a machine code instruction intended for direct transmittal to the card. The format strings are a small array made up in the form:

[0148] FLAG, CLA, INS, P1, P2, LEN

[0149] FLAG:

[0150] if bit 0 set: "write" command

[0151] if bit 1 set: len in parameters must match LEN in string, else error return

[0152] if bit 2 set: P2 in parameters must match P2 in string, else error return

[0153] if bit 3 set: P1 in parameters must match P1 in string, else error return

[0154] if bit 4 set: LEN in string used, parameter ignored

[0155] if bit 5 set: P2 in string used, parameter ignored

[0156] if bit 6 set: P1 in string used, parameter ignored,

[0157] In interfacing with the higher layers, layer 3 uses "command strings". Command strings consist of 1 or more command tags, terminated with a Null. Each command string (CS) is referenced by its CS number or CS name. The same goes for command tags. Command strings can be built from tags and other command strings to perform card related and system related functions.

[0158] Command tags exist to perform almost any possible action within the UCI. In many ways they can be considered as forming a high level programming language. Command tags come in three types, relative, direct and indirect. The behaviour of a direct command tag cannot be varied, the exact behaviour being coded into the tag. Relative and indirect tags use parameters to modify the behaviour of the tag. With relative tags, the modifying parameters follow the tag in the command string. With indirect command tags the parameters are read from a parameter stack, which may be altered by other tags. Examples of command tags include:

[0159] Determine card type & issuer,

[0160] Call Access_Sync_Card function,

[0161] Call Access_Async_Card function,

[0162] Read magnetic card data,

[0163] for card interface matters;

[0164] Send messages to printers, modems, displays,

[0165] Get data from keypads, input switches,

[0166] Read real-time clock,

[0167] for controlling external devices; and

[0168] Copy, manipulate data strings,

[0169] Compare strings,

[0170] Test 'condition',

[0171] Skip next command tag if 'condition',

[0172] Jump to new command string,

[0173] Call new command string,

[0174] for internal program operations.

[0175] For interacting with the cards it is important to know the data structure and format of the card's memory. For organising and supplying the required data regarding the card's memory, the UCI has a file format database. Once the card type and issuer number have been identified, and (if applicable) an application selected, the database can be used to guide access to the applications files. Table 8 is an example of such a database showing the subdivisions through card type, issuer, and application. Files for a given application are referenced by file number, and possible file name, these in turn pointing to a file descriptor.

TABLE 8

Card Type	Issuer	Application	File Number	File Name	File Descriptor
Card A	Bank 1	Bankcard 1	1		
			2		
			3		
			4		
			5		
			6		
			7		
		Creditcard m	1		
			2		
			3		
			4		
			5		
			6		
	Bank 2	Bankcard j	1		
			2		
			3		
			4		
			5		
			6		
			7		
			8		
			9		
			10		
			11		
Card B	Bank 3	Creditcard n			I
					e
					J
					s
		Bankcard k	1		
			2		
			3		
			4		
			5		
			6		
			7		
			8		
			9		
			10		
			11		

[0176] File Descriptors list the following file attributes:

[0177] File type

[0178] Debit key pointer

[0179] Credit key pointer

[0180] Secret code pointer

[0181] Read authority

[0182] Write authority

[0183] Update authority

[0184] File size

[0185] Cypher method.

[0186] Key pointers can also indicate any need for remote key access, AS2805 (or similar) protocols, on-line credit, or other facilities.

[0187] Each time a new file (or "file area" in the case of a memory card) is selected, the file status register is updated. This contains some or all of the following information:

File hierarchy/Directory	
File number/Name	
File identifier	optional long name
File description	purse/keys/standard/proprietary
Option	purse specific
Size	in bytes + bits
Access criteria	read/write/update levels, and locks.

[0188] With cards that do not support the more advanced features, the relevant parts of the register will be blank.

[0189] Similar but more limited information is available from cards which support more than one issuer area. The following is some data typically available when switching directories:

[0190] Directory number

[0191] Access conditions.

[0192] Some general data can be accessed from the card by a rudimentary examination. This information is used by layers 4 and 6 of the UCI to support and corroborate the information derived from Table 8.

[0193] Layer 4 of the UCI program takes care of entity procedures. Its purpose is to interface with the card data structures, files, purses and directory. One of the functions performed by layer 4 is determining the issuer of an inserted card. Once the card type is determined, a number of command strings are trialed to attempt identification of the card file and data layout. An example of command strings to attempt for differing card types are shown in Table 9.

TABLE 9

Card Type	Command String(s) to Attempt	Resultant Issuer Numbers
1	1, 2, 3	16, 16, 16
2	7, 38	3, 8
3	8	2
4	9, 32	15, 8
5	25, 42	16, 9
6	10	16
7	4, 5, 6	9, 16, 16
8	16	12
9	11	4
10	23, 51	3, 4
.	.	.
.	.	.
.	.	.

[0194] If a command string performs as programmed, the resultant issuer number becomes the reference to the card issuer (or co-occupying application).

[0195] Once the card issuer number has been determined, higher code layers can determine the capabilities of the current card and application combination.

[0196] One example of a command string action used primarily for issuer identification is:

[0197] CS n:

[0198] IF card is "Async" and Multi directory

[0199] Select Master directory

[0200] Find First "Unrestricted Read" file

[0201] Open File

[0202] Read File

[0203] IF file contents=CRD pattern

[0204] copy file data to CRD buffer

[0205] set Issuer Number=n.

[0206] Layer 4 also takes responsibility for the key and secret code handling for cards.

[0207] Layer 5 of the UCI contains the process primitives for interacting with the cards. For each card related generic function accessed at layer 8 of the UCI, layer 5 includes a table of card type+issuer numbers showing the corresponding command string for that application. Nulls take the place of command string numbers if the requested function is invalid for the current card or application.

[0208] This table allows translation from the generic function descriptions of layer 8 to the application specific descriptions of the lower layers.

[0209] Layer 5 also includes support routines for:

[0210] Communications protocols (decisions on use of DES, HDLC, modem, deferred uploads)

[0211] Debit, credit transactions

[0212] Financial transactions to AS2805 or similar

[0213] MACing

[0214] Transaction log uploading

[0215] Message formatting

[0216] Security module logical connection to card application

[0217] Key and secret code handling for communications

[0218] RSA cypher.

[0219] While layer 5 includes tables and routines for the process primitives, layer 6 makes use of these tables and routines to translate between the generic functions and instructions of layers 7 and 8, and the lower layers card type or issuer or application specific instructions of the lower layers.

[0220] Layer 7 provides a uniform reduced set of instructions for use by the external device.

[0221] Layer 8 of the UCI isolates the user (the external device) from all the lower levels, so that the user may issue simple generic commands and allow the UCI to take the complicated actions that are required to execute the commands. Effectively the interface at layer 8 becomes seemingly independent of card type.

[0222] The UCI at layer 8 could be communicated with by the user for example by switches, buttons, and relays or RS232 serial data link.

[0223] Examples of the generic commands that would be acted on by layer 8 are:

[0224] New card detected

[0225] Identify new card, read card number, global and user summary

[0226] Account balance (acc)

[0227] Update global, user information

[0228] Debit account (acc, amnt)

[0229] Credit account (acc, amnt)

[0230] Read unformatted data file (filename)

[0231] Update unformatted data file (filename)

[0232] Upload transactions

[0233] Download OS (operating system)

[0234] Read keypad

[0235] Print message

[0236] Display message

[0237] Configure display, printer, modem, I/O devices

[0238] Load key(s).

[0239] As an example of the operation of the UCI, when a new card is inserted, layer 8 receives a signal from "New Card Detected" (layer 1). The external device (user software) may then decide to identify the card, and possibly conduct a transaction.

[0240] From layer 8: execute Identify New Card etc this translates to the following actions:

[0241] execute decide_card_type to determine manufacturer card type

[0242] (this uses procedures covered by tables 2 and 3, these in turn use procedures mentioned in respect of layer 2)

[0243] execute procedures outlined with regard to layer 4 to identify any possible application(s)

[0244] (this uses the functions of layer 3, command strings, command tags, and the related data of tables 6 and 7).

[0245] As another example of the UCI in operation, when debiting an account from a card,

[0246] From layer 8: execute Debit(acc #, amt $) function [L8.1.5]

[0247] From layer 6: use the table in layer 5, check if CommandString exists for "Debit" for this card

[0248] From layer 6: execute CommandString (sequence of Command Tags) if available in table.

[0249] From layer 3: the CommandString is interpreted into following actions:

[0250] select Application Area/subdirectory on card,

[0251] select File or Purse File in subdirectory,

[0252] select correct cypher keys if required (as indicated in CommandString),

[0253] (selection of subdirectories, files and keys is aided by use of Table 8)

[0254] present any cypher keys used,

[0255] alter File or Purse File by "amount",

[0256] record transaction details (if required).

[0257] Therefore the simple generic command "Debit" issued to the UCI is all that is required by the user software, the UCI completing all aspects of the interfacing with the actual card, independent of the card type of issuer.

[0258] The UCI in the preferred embodiment described has the advantages that:

[0259] 1) it can recognise a multitude of card types,

[0260] 2) it knows the features and limitations of each card type,

[0261] 3) it knows the arrangement of data on each card type and all relevant rules for access,

[0262] 4) actions are not predefined, but are determined by external devices, and

[0263] 5) the interface to the external device is simple, so that the manufacturer of the external device needs little or no external knowledge of smartcards, security algorithms, or financial transaction handling.

1. A method of interfacing between a host application program and a data storage card having an associated card application, comprising the steps of:

identifying the particular card application from the card,

selecting from a store of a number of protocols for a number of different card applications the appropriate lower level protocol for the identified card application,

translating high level language host application program commands to corresponding commands within said identified lower level protocol and writing said corresponding commands to said card and translating commands or data in said selected lower level protocol

from said card into corresponding commands or data in said high level language to said host application program.

2. A method according to claim 1 wherein identifying the particular card application includes the steps of reading from the card a data string which characterizes the card application and matching said read data string with a card application-characterizing string from a plurality of stored card application-characterizing strings, corresponding to a plurality of different data storage card applications, to thereby identify the card application.

3. A method as claimed in either claim 1 or claim 2 including the step of identifying high level language host program commands not supported by said card application and upon identifying an unsupported command, returning an error command to said host program.

4. A method of interfacing between a host application program and an electronic data storage card having an associated card application, comprising:

(1) storing first data arrays containing strings of different card application-characterizing data and for each string its corresponding card identifier, and second data arrays containing card identifiers and for each identifier its corresponding command strings,

(2) detecting a card presented to a read/write station communicating with the host application program and passing a card detect signal to said host application program,

(3) reading from the card the data string which characterizes the card application,

(4) searching through at least one said first array for a matching data string, and upon making a successful match retrieving the corresponding card application identifier,

(5) selecting the appropriate command strings in said second array using the retrieved card application identifier,

(6) translating host program instructions to commands appropriate to the card application using the selected command strings from the second array, and

(7) either reading or writing data to said card in accordance with said host program instructions.

5. A method of enabling host application software to interface with any known type of smartcard or other data storage card devices having an associated card application, comprising:

(1) reading from the card to be interfaced the data string which characterizes a particular card application,

(2) searching through a first data array containing strings of different card application-characterising data, and for each string its corresponding card application identifier, for a matching data string, and upon making a successful match retrieving the corresponding card identifier,

(3) selecting the appropriate command strings from a second data array containing card application identifiers, and for each identifier its corresponding command strings, using the retrieved card application identifier,

(4) translating generic host application instructions to lower level commands appropriate to the card application being interfaced using the selected command strings from said second data array, and

(5) causing data to be either read from or written to said card in accordance with said generic instructions.

* * * * *