Edgar H. Sibley
Panel Editor

In the Arab world the need for bilingual microcomputer systems is ever increasing. In addition to the ability to process the Arabic and English scripts, an ideal system should support the use of existing applications with Arabic data and the access to the system facilities through Arabic interfaces. The Integrated Arabic System (IAS) was developed to study the feasibility of building such systems using existing microcomputers and software solutions.

BUILDING BILINGUAL MICROCOMPUTER SYSTEMS

Murat Tayli and Abdulla I. Al-Salamah

Arabic, one of the five official languages of the United Nations, is the mother tongue of some 200 million people in 21 countries [6]. The Arabic alphabet and script are also used, with minor variations, to write other languages like Farsi, Urdu, and Malay markedly extending the geographical and economical impact of this language. In all of these countries, a wide range of computing tools, originally designed to process Latin-based scripts, have been in use for decades despite the linguistic barriers and inadequacy of their user interfaces.

In addition to the need for ad hoc peripherals capable of handling Arabic script, the computer profession requires, at almost all its levels, an additional qualification: a proficiency in the English language. This added burden restricts many who would benefit from computers if the facilities were available to them in their own language, particularly at entry levels. One can easily imagine the potential problems if computer operators, in the United States for example, had to control their machines by using Japanese commands coded with Kanji characters. In many countries, the investment in linguistic training and the use of specialized input/output units have been relatively successful when the operation of computing tools have been isolated from the production environment. For instance, many organizations equipped with mainframe systems have managed to serve large user communities with a relatively limited number of well-trained specialists. The advent of microcomputer systems on the market, however, has radically altered this remote man-machine relationship and placed the end-users in the front line.

To meet the pressing needs of the Arab market, several companies that operate mainly in the Middle East, have developed a wide range of products for available microcomputer systems. In addition to Arabic versions of popular business-oriented applications, a number of software tools and utility programs [1, 8, 14, 15, 18] have been provided to process Arabic and bilingual data with standard input/output units of these systems.

Although all these products helped to boost popular usage, the language problem remained largely unsolved. The installation and operation of such products still required the knowledge of the English language. Furthermore, the case-by-case search for particular solutions resulted in the proliferation of different norms that often conflicted with each other and led to the severe incompatibility problems. Whatever their scope, all the efforts invested to incorporate the Arabic language in computer systems have been commonly referred to as *the arabization of computer systems* or in its abbreviated form the *arabization* process. The contextual semantics of this term is redefined in various parts of the article.

This article presents the design principles and architecture of a bilingual system called the *Integrated Arabic System*. This experimental system, built around the IBM Personal Computer family, is also used as a framework to introduce and discuss problem areas of the arabization process at the workstation, operating system, and application levels.

MOTIVATIONS AND OBJECTIVES

This study has been designed to answer the following question:

> Can an existing microcomputer system, designed to operate in Latin-based environments, be transformed to an Arabic system, with bilingual script-processing capabilities and still keep most of its original assets?

If an affirmative answer is the aim, the objectives that must be achieved include:

- the building of a system which operates in Arabic and which supports bilingual (Arabic-English) script-processing capabilities, and in which the linguistic

issues are treated with a coherent and unified set of solutions;
- the provision of a series of development tools, whereby those interested could rapidly build new Arabic and bilingual applications;
- the usage of existing applications, designed to process Latin-based scripts, with Arabic and bilingual data;
- the operation and usage of existing Latin-based packages, without any modification or restriction.

DESIGN GUIDELINES

To delineate the framework of the bilingual system a number of design guidelines, (depending on more or less heuristic considerations) were set. These were:

- for economic reasons, the arabization process had to rely mainly on software solutions and let the user exploit basic capabilities of the system without restriction;
- to address a large audience, the core of the bilingual system had to be a popular and relatively powerful workstation;
- to dissociate hardware dependent issues from higher abstraction levels (operating system and application layers), problems inherent to the arabization of input/output functions had to be solved, whenever possible, at the workstation level;
- to support complex layout design requirements of the display unit and allow its sharing, basic capabilities of the workstation had to include advanced display management and resource multiplexing mechanisms;
- to provide a complete Arabic processing environment, the operating system of the workstation had to support Arabic user interfaces; and
- to meet the functional objectives, the final system had to include bilingual development tools and programming interfaces to existing languages, at its application layer.

PROBLEMS OF BILINGUAL TYPESETTING

The initial phase in building a bilingual system is to look for appropriate solutions to represent selected languages accurately on the target computing media and secure their coexistence with minimal compromise. In the context we define, problems to overcome fall into three categories:

(1) issues related to the representation of the Arabic script;
(2) inconsistencies resulting from typesetting mixed language layouts;
(3) selection or definition of an appropriate bilingual character code set that satisfies both international standards and requirements of the specified system.

Typesetting of the Arabic script on computing media originally designed to process Latin texts constitutes the first challenging step in the arabization process. Difficulties stem not only from fundamental dissimilarities between the Arabic and Latin scripts, but also from the lack of well-established standards for the automation of Arabic typesetting. Problems of automatic processing of the Arabic script and bilingual typesetting are thoroughly covered in [5] and [12]. The following sections summarize characteristics of the Arabic script and highlight problem areas of the arabization process.

Characteristics of Arabic Script

To the uninitiated, Arabic script exhibits a number of unusual and confusing features. The basic alphabet consists of a set of 28 letters, representing mainly the consonants and a few long vowels. It is extended to some 90 elements by additional shapes, marks, and vowels formally recognized in the Arabic morphology. Arabic letters, written in elegant cursive forms, do not differentiate between upper and lower case figures (Figure 1a).

A large number of diacritical signs, which are similar to accent marks of European languages, are used to

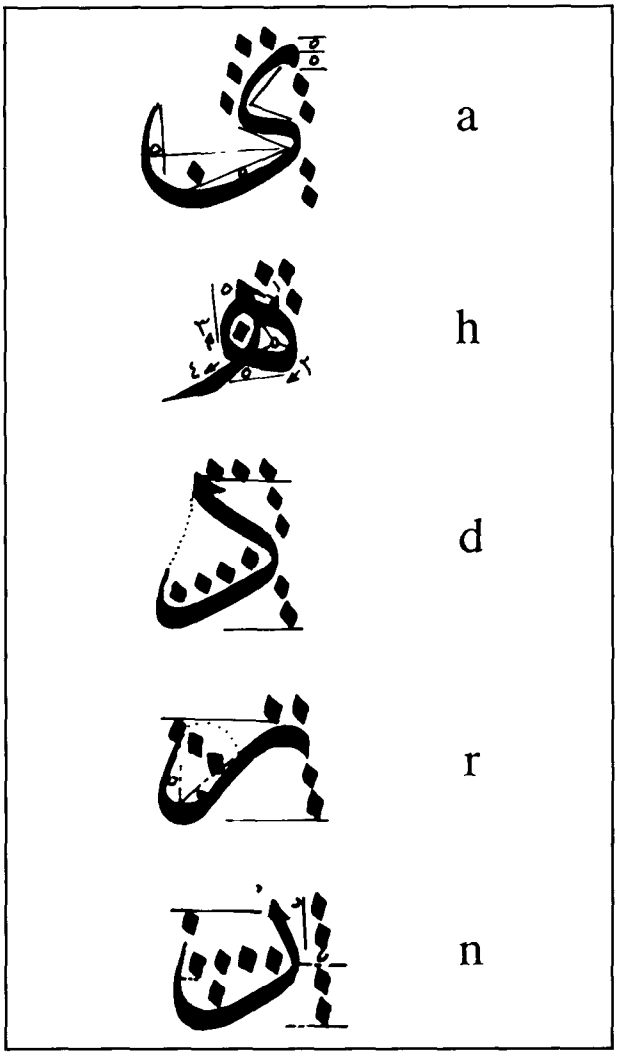

Figure 1a. A sample of 5 letters displays cursive forms adopted in Arabic Calligraphy. Frames and pen strokes placed around characters mark the slopes and scaling factors.

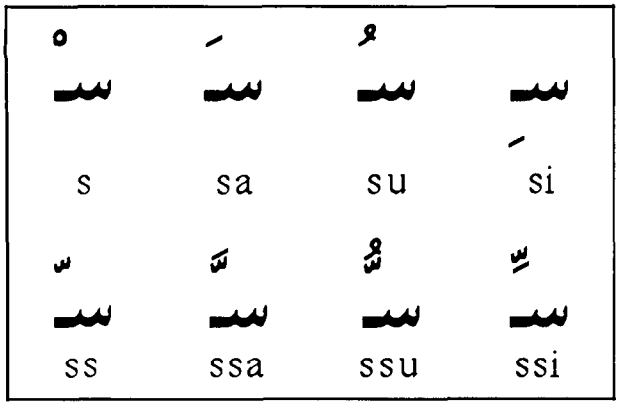

Figure 1b. Phonetic value of basic elements of the Arabic alphabet are extended by a rich collection of diacritical marks. The letter "S" is shown in a number of configurations altering its sound.

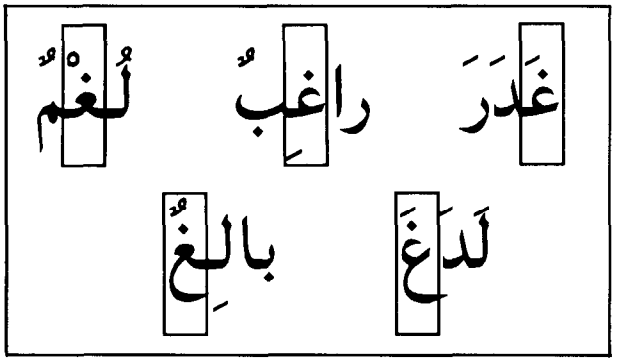

Figure 1c. Shapes of Arabic letters depend on their context. Rectangular frames mark various shapes of the Arabic letter "G", when used at the beginning, in the middle, and at the end of a word.

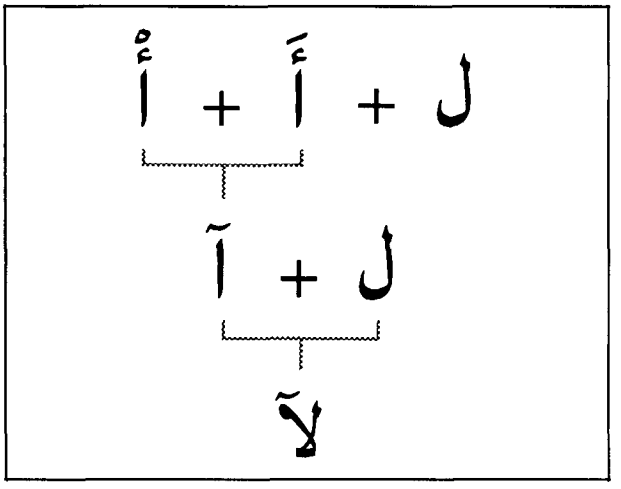

Figure 1d. Some Arabic letters can be fused to form new shapes. A compulsory case is that of three letters (from right to left): the Lam, Hamza-Fathah, and Hamza-Sukoun generating the shape the Lam-Mad character.

mark short vowels and emphasize or loosen a letter's sound. These marks can be mixed and written above or below the characters to produce composite phonetic effects (Figure 1b).

Arabic script, which evolved in contemplation of the traditions of handwriting, is context sensitive. The shape of most of the characters depends on their position within a word and the characters adjacent to them. Each character may be represented up to four different ways of which only one would be correct in a particular situation (Figure 1c). Moreover, many of these intermediary forms depend on the adopted calligraphic style. As an extreme case of contextual reshaping, Arabic script allows ligatures between characters. In other words, adjacent letters can be fused to produce new graphical forms. In order to emulate handwriting, approximately 400 ligatures are available, yet only three of them have mandatory usage (Figure 1d).

Orientation of the Arabic script is from right to left, but Arabic numerals are written and read from left to right (Figure 1e). Unlike Latin-based alphabets, elements of the Arabic script convey directional semantics, which control the orientation of a typesetting process (a more detailed discussion is presented in the coming sections).

Arabic calligraphy adopted a sloping and curved concatenation model as opposed to the flat baseline of Latin-based scripts. This flowing style, along with diacritical marks, contextual shapes, and ligatures, form essential decorative elements of the Arabic handwriting. Nevertheless, the resolution of conventional computer display media is often a limiting factor that prevents accurate reproduction of complex forms. Automatic typesetting of the Arabic script on nonspecialized equipment becomes, rather, an approximation of an original calligraphic style (Figure 1f).

Typesetting of Mixed Scripts

Arabic and Latin-based scripts have opposite directionalities. A composite text, including a mixture of these scripts, can be typeset in two different ways considering one or the other language as the host context. The type of the host, which imposes the default orientation of the layout, is to be fixed explicitly by external agents when no implicit indication is available. Elements of the guest script are treated as counterflow insertions to the mainstream. Punctuations and non-alphabetical characters common to both types of script can cause contextual confusions and contribute to the complexity of bilingual typesetting.

Typesetting of mixed scripts is somewhat analogous to bidirectional typesetting of Arabic literals and numerals. Provisions can be made to try to handle both cases with a common set of mechanisms. Nevertheless, mixed typesetting maybe a recursive process and become intractable when nested insertions occurs, i.e., the insertion of an Arabic text that includes some Latin script, within a Latin host.

The basic constraint in the automation of mixed script typesetting is the exclusion of all solutions that

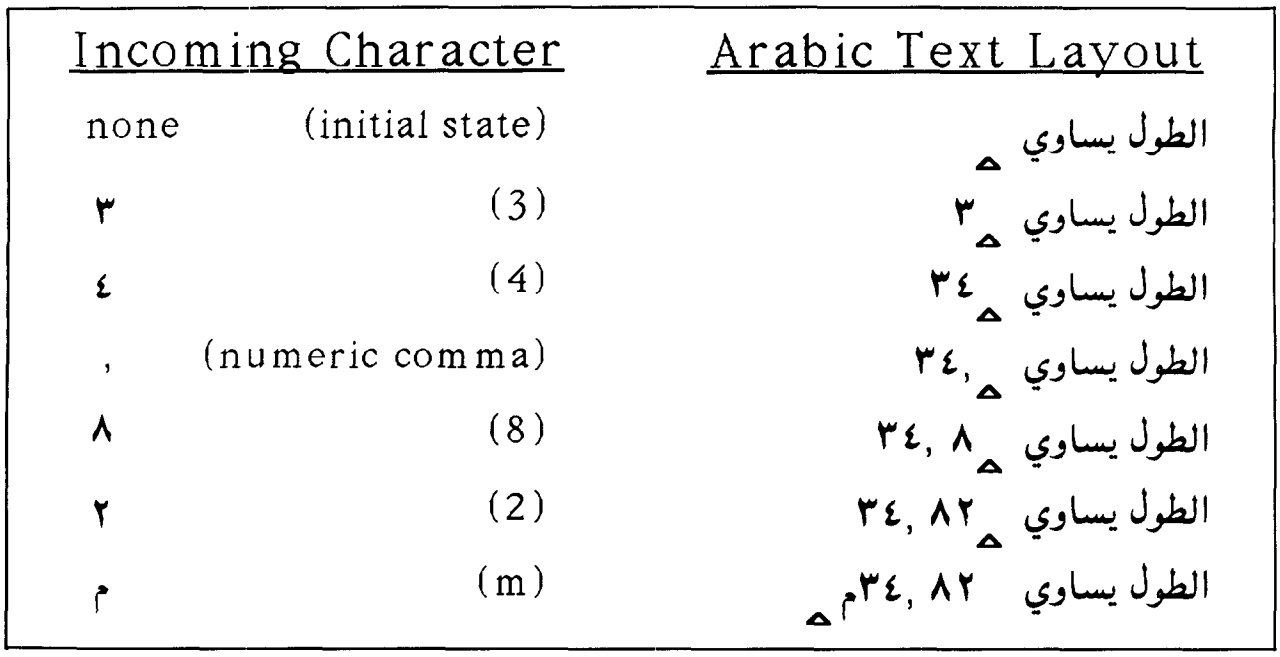

Figure 1e. Arabic literals and numerals have opposite directionalities. Orientation of the Arabic script is from right to left, but numerals are written and read from left to right. Triangular mark visualizes position of the cursor, and its movements on the advent of various types of characters.

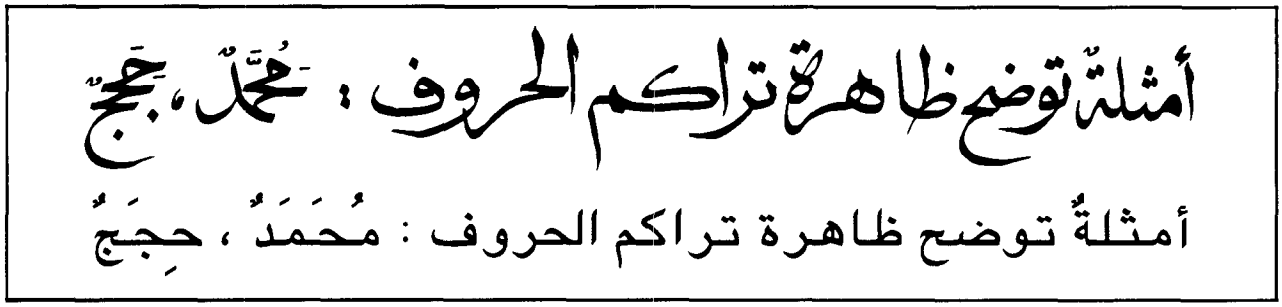

Figure 1f. Representation of Arabic script on conventional computer media is often an approximation of a calligraphic style. Differences between handwriting and printer output become especially noticeable when adjacent letters can be stacked on top of each other.

use special formatting characters along with the original text. In other words, in order to guarantee the portability of existing software, decisions on the direction of the layout have to rely only on the semantics of character codes, not the presence of extra directional idiosyncrasies in the stored text.

Bilingual Character Set

The character set is one of the elementary, yet critical, features of a computer system. Its ability to represent the language correctly and be accepted readily by users are primary criteria to measure the adequacy of its design. To the designers from English speaking countries, the availability of well-established international standards, such as the ASCII and EBCDIC codes, imposes de facto norms and eliminates the need to consider the definition of new sets. Where languages are written with Latin-based scripts, slightly modified versions of these codes can be easily provided at the cost of minor compromises. Even where non-Latin scripts are used, provisions have been made to fit national code sets within these 7 or 8 bits frames in order to ensure the portability of existing hardware and software components [8]. Nevertheless, more complex coding schemas are necessary to portray several Middle Eastern and Asiatic languages accurately. For example, the use of extended coding schemas has been recommended to censure the originality of the Chinese and Arabic calligraphies [4, 13, 17]. Consequently, for the designers from these countries, the decision to adopt an international character code standard or define a better national set determines whether existing products may or may not be integrally used.

ARCHITECTURE OF THE INTEGRATED ARABIC SYSTEM

The Integrated Arabic System (IAS) is organized around a kernel consisting of an arabized workstation called

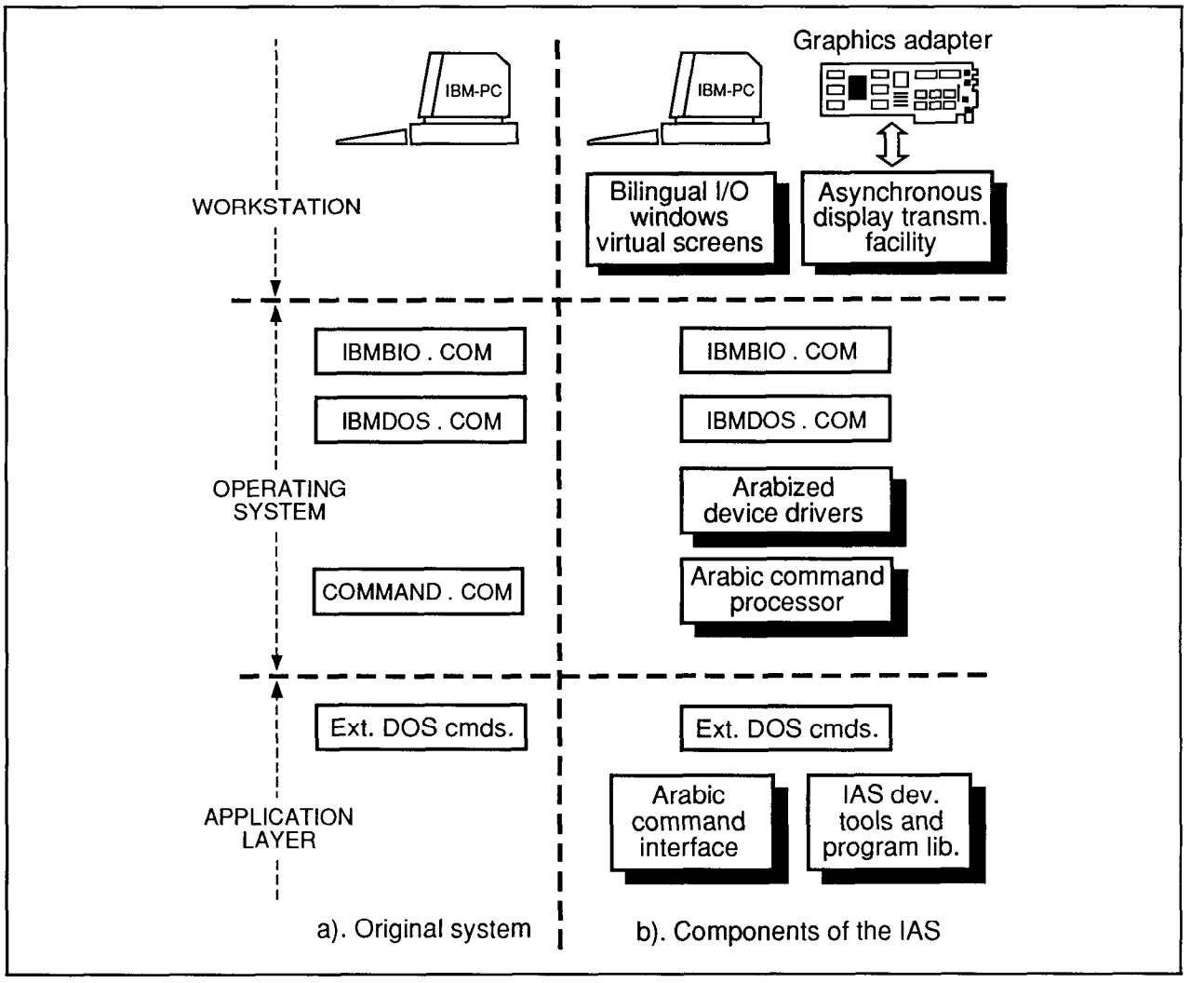

Figure 2. Architecture of the INTEGRATED ARABIC SYSTEM

the Intelligent Arabic Workstation (IAW) and the Arabized Operating System (AOS). It also includes a series of tools, facilities, and interfaces to assist users in the development of new applications and the arabization of existing ones. Figure 2 depicts IAS architecture as compared to the reference system. Shaded boxes represent new modules implementing bilingual processing functions and extensions carried out on the original system to augment its basic capabilities.

INTELLIGENT ARABIC WORKSTATION

The IBM Personal Computer, equipped with an Enhanced Graphics Adaptor (EGA) [9], was selected to be the target workstation. The rationale of this adoption depended mainly on IAS's design guidelines. First of all, the IBM-PC is a popular and widely available machine with numerous clones. A relatively large number of models exists that correspond to various systems with different power. Furthermore, it has an open architecture and can be used as a building block to form more complex systems through various networking strategies.

Conforming to our design principles, arabization of most of the input/output functions has been carried out at the workstation level. In addition to linguistic extensions, the Basic Input Output System (BIOS) of the workstation has been modified to redefine the video display unit as a logical and multiplexible resource. Furthermore, an alternative display typesetting mechanism, the Asynchronous Display Transformation Facility (ADTF), has been added to the workstation's arsenal in order to meet special layout requirements.

Bilingual Character Set

The IAW has been provided with a nonstandard, 8-bit, ASCII-like bilingual character set [19]. The new set was designed to comply with IAS's objectives, as existing national [2, 3] and international standards [10, 11] did not satisfactorily meet the specified requirements.

In the new set, Latin characters keep their original

positions in the first half of the codes [8]. The second half, codes from 128 to 255, is devoted to the Arabic alphabet and its associated shapes. A sufficient number of diacritical marks is included in the set to improve the accuracy of the Arabic script representation.

Two special codes, the numeric space and the numeric comma (traditionally used as the Arabic equivalent of the decimal point) have also been added to the set in order to ensure proper visualization of numerals and Arabic literals including numeric characters. This double coding for the comma and space characters is due to both the bidirectional orientation of the Arabic script and our decision not to include directional idiosyncrasies in the stored text. For example, the string "869_7440" represents either two distinct numbers separated by a space in Latin-based scripts or a numeral formatted in a special way, such as a telephone number. The right semantics of the string is deduced from the context in which it occurs. As for the Arabic, the numeral has the same representation as its Latin counterpart, but numbers 869 and 7440, which are separated by a space, are written as "7440_869", given the right-to-left orientation of its script. As the same series of characters, "8-6-9-_-7-4-4-0," is presented to the system for both cases, only the semantics of the space character can be used to decide on the correct Arabic layout. Similar considerations are also valid for a number of other separators such as the comma, hyphen (also used to represent the minus sign), parentheses, and brackets.

Bilingual Typesetting

IAW adopted Arabic as the host language, setting the native direction of the output media from right to left. Latin characters and Arabic numerals, gathered in a single subset, are treated in the insertion mode with no provisions for nested operations. Most of the punctuation and non-alphabetical characters form a neutral subset, thus having no impact on the orientation of the layout. Variable shapes of Arabic characters are dynamically generated through a context analysis by substituting provided codes with their proper alternatives. In addition, the screen layout is automatically readjusted to render the correct typesetting. For instance, to preserve the logical orientation of the line and mark the position of the next character on the display, the cursor is set: to the left of an Arabic literal or to the right of a numeral or a Latin string. In the latter case, the advent of an Arabic character causes the cursor to skip over the entire string and be placed to the left end of the current line (Figure 1e). Conversely, a destructive backspace function is expected to restore the previous visual effect.

The above paradigm along with the adopted character set established a workable typesetting model in which the use of extra directional codes is avoided. Users are provided with a transparent working environment where they feed their input to the system in the natural order and where stored data is processed and transmitted in the chronological sequence [19]. Nevertheless, IAS also requires the definition of an additional typesetting schema in order to run off-the-shelf products without modification and use them with Arabic data. This complementary model, which has to handle composite scripts and a mixture of host contexts, is left to the control of the unorthodox ADTF mechanism presented in the following sections.

Advanced Display Management Mechanisms

The interactive nature of the IAS environment evoked the idea that the extension of IAW's capabilities by a number of elaborated display management mechanisms would be a judicious investment to promote future developments. Observations revealed that designers often face two categories of problems in the definition of their screen layouts:

- the restrictions imposed by the physical limits of the output media, and
- the need for dynamic formatting of displayed frames.

Similar problems also exist at the operating system level and are emphasized by the need to create multiple instances of this non-sharable resource. In order to meet the requirements of both the operating system and application layers, capabilities of the IAW have been extended to support two new objects: the virtual screens and the windows. The semantics of these complementary objects and visual effects resulting from their simultaneous use are sketched in Figure 3.

The virtual screen mechanism allows the creation and concurrent use of a number of logical screens whose dimensions may exceed physical limits of the display unit. A given application is allowed to create, a priori, an unlimited number of virtual screens (in reality limited by the available memory) and write simultaneously on all of them. Nevertheless, only one of the virtual screens can be displayed at a time. Based on the position of the cursor and its subsequent movements, the selected virtual screen is mapped on the physical unit. If the cursor reaches the limits of the output media, the view is automatically reframed by a number of columns or lines in the indicated direction.

Dynamic formatting of the display unit is carried out by better-known objects: the windows. As opposed to virtual screens, windows are assigned to fixed coordinates on the physical unit and keep their position until they are cancelled. Their size might be equal to or less than the display unit. Like virtual screens, any number of them can be created and used simultaneously, subject to the same memory restrictions. Unlike virtual screens, all the windows are concurrently displayed, covering each other fully or partially. The visibility of the windows is controlled by a simple mechanism, which queues the reference of created objects in a system list and builds the screen image by overlapping listed items in sequence. The resulting view becomes, by default, a pile of windows stacked in chronological order. The reference list is, however, a global IAW object. It can be accessed by the rest of the system and reorganized at will. The operating system and applications have the liberty to define their own window hier-

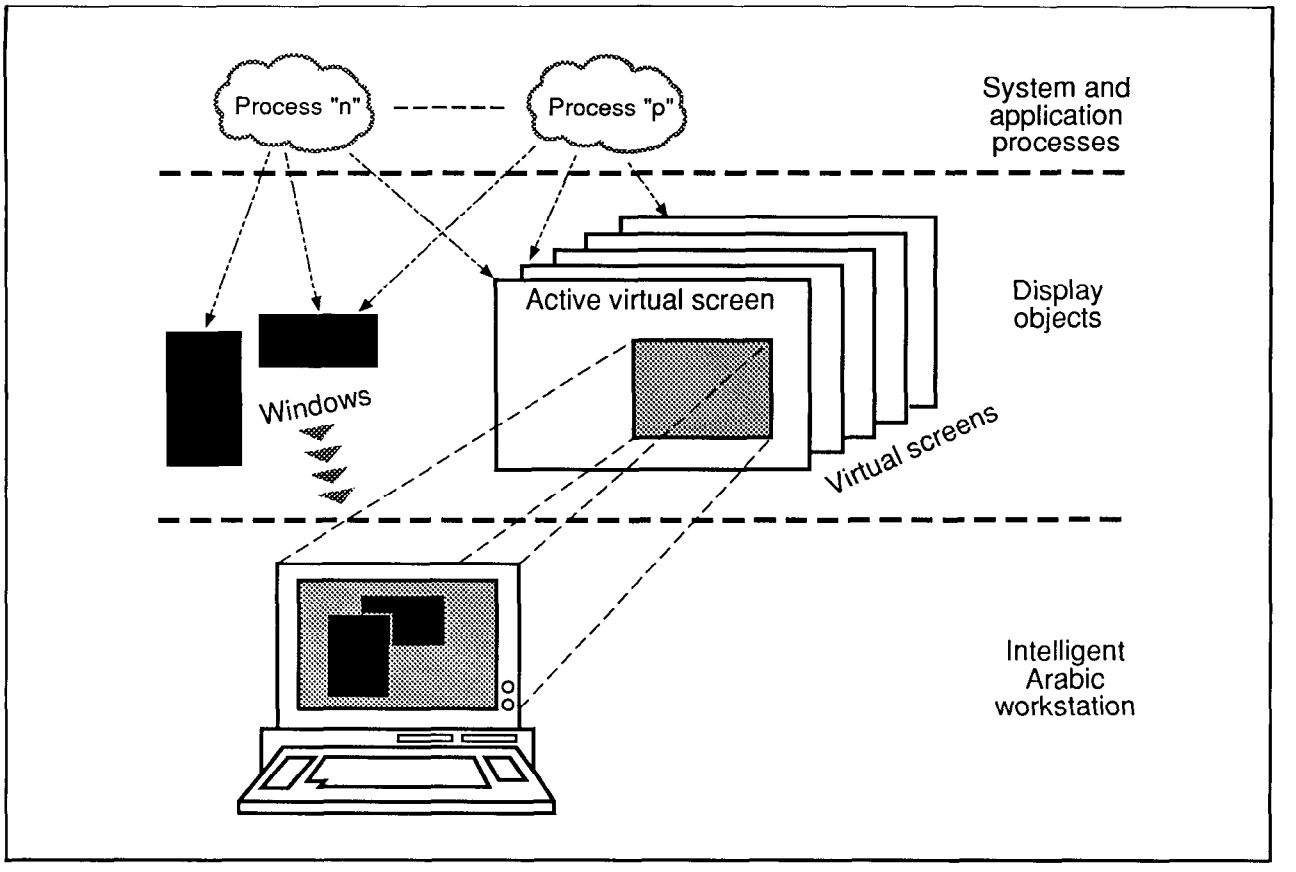

Figure 3. Hierarchy of Virtual Screens and Windows

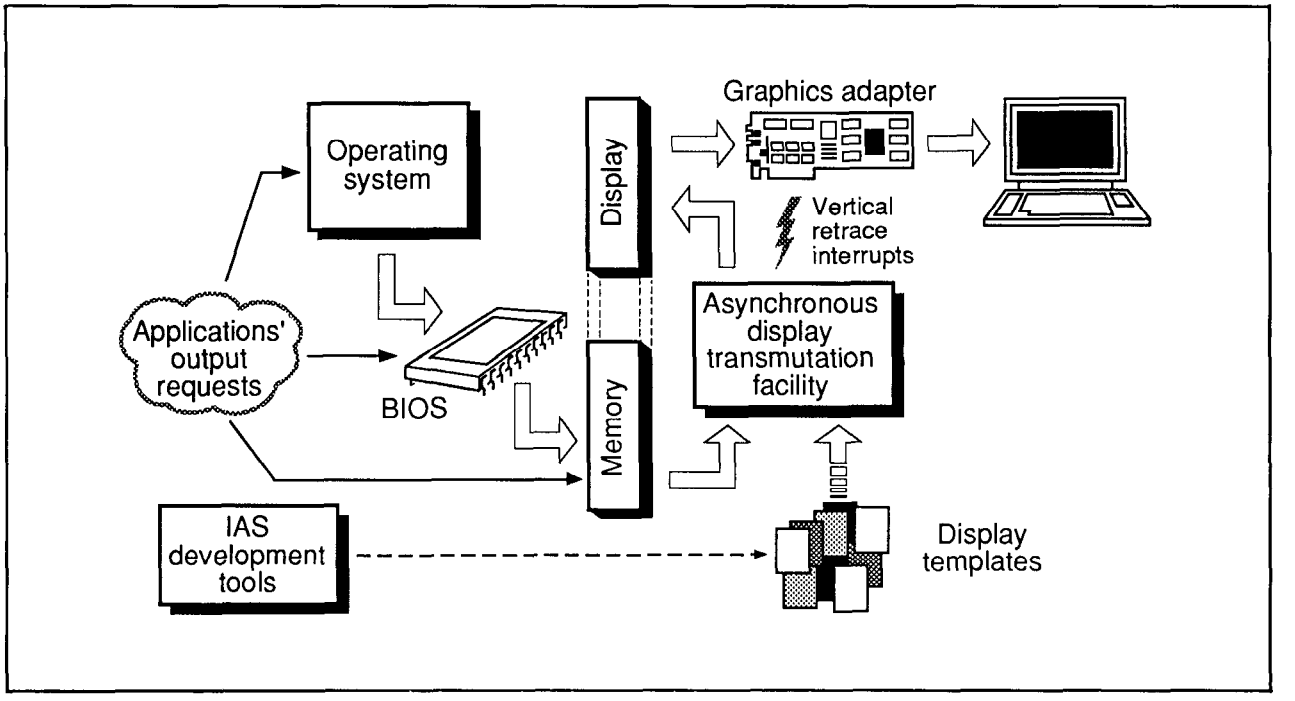

Figure 4. Logic of the Asynchronous Display Transmutation Facility

archy such as trees and circular queues by reordering the reference list and obtain automatically corresponding visual effects.

IAW's support for the virtual screens and windows is kept limited to basic control functions and text-editing primitives common to both categories [21]. At the initialization of the system, IAW redefines the display unit as the default virtual screen, with a permanent life span. Other display objects are created, activated, and destroyed on request as needed by the operating system and running applications. Processes access and share display objects through unique identifiers and can ask for the redirection of output requests toward other objects. IAW does not impose a priori policies on the utilization of display objects. Decisions such as: how much of the screen is devoted to a window; how and when an object is resized; whether windows overlap or are laid out as tiles; and where keystrokes are directed, are relegated to higher levels such as "Screen or Presentation Managers" or specialized applications.

Asynchronous Display Transmutation Facility

IAW's typesetting mechanisms described so far, guarantee the arabization and proper functioning of most of the existing applications. Nevertheless, a number of packages would still produce incorrect or scrambled layouts when used with Arabic data. They correspond to a category, which can be qualified as irregular, accessing the display memory directly and bypassing all standard system interfaces. Consequently, as neither the operating system nor the BIOS control the flow of the operations, arabization procedures integrated to them are also ineffective as shown in Figure 4. The ADTF is used for the arabization of such packages and the control of special typesetting needs that are not supported by IAW's native mechanisms. One such special case is the mixture of Arabic and English scripts, both used as host contexts in the same screen layout. The situation occurs when off-the-shelf, Latin-based applications such as word processors or database management systems, which usually format the screen in data and command fields, are used with Arabic data. A reasonable user expectation is to see command fields kept in their original form and data typeset in Arabic. ADTF's basic function is to transform displayed images according to those specific layouts, without being noticed either by the application or the user.

ADTF is implemented as part of EGA services and operates asynchronously from the rest of the processes in the system. It is periodically activated by the interrupts of the EGA that, as Figure 4 shows, are generated at the end of each vertical retrace cycle. Once operational, ADTF periodically scans the display memory, rearranges, and typesets its contents according to a prescribed layout. The implementation of the ADTF, as a phantom process at IAW level, provides a transparent and application-independent mechanism. Another integration policy, such as its inclusion at the application level or in the operating system, would have failed to guarantee the same degree of reliability and transparency. For example, the scheduling of the ADTF activities would have depended on the system clock (a heavily used system resource) and caused a potential threat to critical time-dependent operations. Moreover, asynchronous accesses by the ADTF and EGA to the display memory also would have produced flickering and snowing effects on a number of video units [9].

ARABIZED OPERATING SYSTEM

The unusual term "Arabized Operating System," which we use, refers to a system that:

- interacts with the users through Arabic interfaces,
- identifies system objects, i.e., logical and physical units, directories, and files, with Arabic names, and
- implements bilingual script processing capabilities on various input/output devices.

IBM-PC DOS, the de facto market standard for this line of personal computers, was selected to form the core of the AOS. Its adoption depended on a series of considerations, similar to those involved in the selection of the workstation. The segmented architecture of DOS shown in Figure 2a helped to isolate problem areas and ease the development process. The loading sequence of DOS modules in the memory corresponds somewhat to abstraction levels of the objects in this system. The first module that is loaded at system initialization (IBMBIO.COM) consists of low-level functions to interface the physical machine and the BIOS. It is followed by the file management system (IBMDOS.COM) and the command processor (COMMAND.COM), which carries out the majority of user dialogues. A series of independent utilities, called "external DOS commands," complete the operational environment of IBM-PC DOS.

The arabization of IBM-PC DOS relied basically on the guidelines and procedures defined in this system for the implementation of customized solutions [7]. It consisted of the development of a series of device drivers and a new command processor as shown in Figure 2b. External DOS commands have been treated as special application programs and arabized by means of similar mechanisms.

Bilingual Device Drivers

Architecture of IBM-PC DOS includes provisions for the installation of user-defined device drivers. AOS exploits this feature to implement two different services. First, it installs new drivers that control input/output units according to the semantics of bilingual typesetting rules. Second, it names logical and physical system objects such as line printers, system consol unit, and input/output ports in Arabic [20].

IAW's display management mechanisms provide substantial support in the development of bilingual device drivers, especially for output units. The simulation of device data buffers by means of virtual screens lets IAW's input/output primitives handle the complex bilingual typesetting task. For example, AOS printer drivers consist of two layers. A generic layer, common to all

printer types, creates a virtual screen for each printer. It has the dimensions of one line and a number of columns representing the width of the printer. Outputs to printers are first directed to associated virtual screens for bilingual typesetting. A second layer, formed by device dependent modules, maps the contents of these virtual screens on corresponding physical devices.

In addition to Arabic mnemonics, AOS also preserves original English denominations of system objects. This dual naming schema helps to assign different behavioral patterns for some devices. For example, a printer equipped with a proper bilingual character set produces outputs according to the Arabic layout when addressed to by its Arabic name and acts in its native mode (Latin) when the corresponding English mnemonic is used.

Arabic Command Processor

The user interface of IBM-PC DOS resides in its command processor. In AOS, this processor is replaced by a fully compatible and functionally equivalent module, the *Arabic Command Processor* (ACP). The ACP accepts commands entered in Arabic or in English, but replies exclusively in Arabic. As with the original module, multiple copies of the ACP can be activated simultaneously or invoked from application programs. ACP can also be interleaved with the original command processor when necessary [20].

The major functional difference between the two command modules resides in their memory utilization policy. While the ACP is a single-resident module following other DOS segments in the memory, the original command processor splits itself into two segments. A small resident segment relocates the rest of the code to the other end of the memory. This larger segment, transferred in the user area, is occasionally overwritten by the running applications. The decision to implement the ACP as a single resident segment depended on obvious technical considerations. First of all, a transient segment of some 20Kb does not constitute a substantial memory saving, given the large memory configurations of today's microcomputers. Further, the necessity to reload a frequently used system segment is a cumbersome and time consuming process. Finally, the use of a nonreentrant module is a major drawback when multiple copies of the command processor have to be invoked simultaneously.

APPLICATION LAYER

The IAS application layer includes a number of programming tools, system utilities and arabized versions of the most frequently used external DOS commands. The development of new applications is supported by a series of program libraries containing system interface routines for Pascal, C, and Macro Assembler languages [21]. An Arabic line editor helps in the preparation of bilingual text files and elementary documents.

A screen layout design utility is provided to prepare user-defined templates for the ADTF mechanism (Figure 4). This facility is particularly useful when a given screen layout requires the mixture of both the Arabic and Latin scripts as host contexts. For example, a spreadsheet or a word processing program arabized with the help of ADTF would provide, by default, reversed screen images suitable to the Arabic script, in which the upper left corner of the image is mapped to the upper right corner of the screen and vice versa. Yet, parts of these images, such as header and trailer lines, and command menus are reserved to system messages and are likely to be coded in English. To keep the original orientation and formatting of these areas, ADTF must be guided with specific, case-dependent templates, which are dynamically associated with the current application. In its simplest form, a template consists of a boolean array that defines the native orientation of the lines on the screen. ADTF refers to this vector to determine the host context and lessen the ambiguities inherent in the bidirectional typesetting of mixed scripts (as presented in earlier sections). In practice, it has been shown that most applications require only a limited number of such templates. Once generated, these can be filed for future use. The association of templates with the applications is done in a number of ways. For frequently used packages, the layout design facility generates an encapsulation code to load the concerned template and start the selected program. Users run their application by invoking encapsulation codes. A template can also be explicitly loaded using appropriate system commands. For casual cases, a hotkey service allows users to display and change the current template interactively.

The Arabic Operating System is a functional replicate of DOS. As such, it does not present a more user friendly interface than the original system. Users, despite their interaction with the system in Arabic, still have to deal with the line-oriented syntax of DOS commands and their even less obvious parameters. In an effort to reduce the syntactical diversity of system commands and assist non-sophisticated users, IAS was equipped with an optional system interface: the Arabic Command Interface (ACI). ACI aims to ease operation of the system through the use of single key stroke actions, appropriate command menus, and a display of current information on the operational context [20]. Unlike other user-defined shells it is tightly integrated with the underlying system. To execute a command, which requires the support of the operating system, ACI passes the request to a resident command processor instead of invoking another copy of it. The ACI represents a typical example of an application that exploits the facilities offered by IAW's display management mechanisms to fullest extent. The main menu is a virtual screen on top of which command dependent windows are popped. Before transferring the control to the selected program, ACI activates the default virtual screen, making it possible for the application to run in the original context and create others. At the completion of the execution, ACI regains the control and resumes its operations by restoring its initial virtual screen. Like other applications, this facility may be dis-

continued at will, leaving the user in direct contact with the native operating system interface.

CONCLUDING REMARKS

The IAS became operational at the end of 1986, and about one hundred copies have been distributed for assessment. Hardware requirements of the IAS remain modest and limited to a display adaptor with a downloadable character generator and some 70Kb of memory for resident parts of the system. The performance of the system and especially that of the workstation did not show measurable degradation despite the complexity of the bilingual typesetting algorithms and sophisticated display management mechanisms.

The IAS project has shown that it is feasible to build bilingual systems using currently available microcomputers and software solutions. Although English and Arabic were considered in this investigation, the concepts are applicable to other language pairs that do not necessarily belong to the same linguistic family. The conversion that allowed the microcomputer system to operate in a language other than and quite different from the one for which it had been designed was further complicated by pragmatic considerations. First, there was a desire to preserve the native assets of the original system. Moreover, the necessity to port existing applications on the new system and use them with either language or a mixture of the two created a number of additional challenges. IAS's layered architecture helped to overcome many of these issues. Thus, for example, the problems encountered in the design of common linguistic interfaces to output media, which result from the use of different alphabets (and, therefore, different character sets) have been contained at the workstation level. Similarly, the differences in typesetting rules (including orientation of the script, ligatures, and diacritical marks) have also been solved at this level. The operating system was mainly involved in the translation process, and components at the application layer provided the necessary logistic support for the new environment.

In retrospect, the fundamental problems faced during the IAS project resulted mainly from:

- the inadequacy of existing bilingual character code sets (including the one adopted by the IAS) to accurately represent the Arabic language,
- the lack of appropriate bilingual keyboard layout standards, and
- the quasi-absence of proper Arabic computer terminology and the nonstandard nature of current technical vocabulary [16], issues that have been conscientiously kept beyond the scope of the investigation.

REFERENCES

1. Appropriate Technology Ltd. *APTEC Arabic Utilities Programming Manual v2.* London, 1984.
2. Arab Standards and Metrology Organization. Data processing 7-bit coded Arabic character set for information interchange. ASMO 449 Tech. Rep., Amman Jordan, 1982.
3. Arab Standards and Metrology Organization. 8-bit coded Arabic/Latin character set for information interchange. ASMO DS 708 Tech. Rep., 1985.
4. Archer, N. P., et al. A Chinese-English microcomputer system. *Commun. ACM 31,* 8 (Aug. 1988), 977–982.
5. Becker, J. D. Arabic word processing. *Commun. ACM 30,* 7 (July 1987), 600–610.
6. Dempsey, M. W. *Atlas of the Arab World.* Facts on File Pub., New York, 1983.
7. International Business Machines, Inc. *DOS Technical Reference v3.1. No. 6138536.* 1985.
8. International Business Machines, Inc. *Personal Computer National Supplement Arabic. No. 8223445.* 1985
9. International Business Machines, Inc. *EGA Technical Reference: Options and Adaptors. No. SS34-000700.* 1986.
10. International Organization for Standardization. Information processing—Arabic 7-bit coded character set for information interchange. ISO/TC 97, ISO 9036 Tech. Rep, 1987.
11. International Organization for Standardization. Information processing—8-bit single-byte coded graphic character sets—Part 6: Latin/Arabic alphabet. ISO/TC 97, ISO 8859-6 0, Tech. Rep., 1987.
12. MacKay, P. Typesetting problem scripts. *Byte 11,* 2 (Feb. 1986), 201–218.
13. Manzer, M., and Mahmoud, N. N. A special purpose computer for Arabic text processing. In *Proceedings of Tenth National Computer Conference* (Jeddah, Saudi Arabia, Feb. 28–March 2). 1988, pp. 679–688.
14. Microsoft *MSDOS User's Guide Arabic Supplement.* Microsoft, 1988.
15. O1 Systems *ITHA User's Guide and Advanced Programmers Reference Rel. 3.0.* Bahrain, 1988.
16. Saad, H., Adnan, N., and Mohammed S. Recent experience in the development of a standard Arabic information processing glossary. In *Proceedings of Eleventh National Computer Conference* (Dharan, Saudi Arabia, Mar. 4–7). 1989, pp. 373–381.
17. Sakamura, K. BTRON: The business-oriented operating system. *IEEE Micro,* 7 (Apr. 1987), 53–65.
18. Saudi Soft. *AL MUSSAED AL ARABI.* Jeddah, Saudi Arabia, 1986.
19. Tayli, M., and Nafisah, M. Intelligent Arabic workstation. In *Proceedings of the Ninth National Computer Conference* (Riyadh, Saudi Arabia, Sept. 23–27). 1986, pp. 10-2-1 to 10-2-8.
20. Tayli, M. Integrated Arabic system. In *Proceedings of the First KSU Symposium on Computer Arabization* (Riyadh, Saudi Arabia, Apr. 6–9). 1987, pp. 135–143.
21. Tayli, M. Integrated Arabic system technical information and programming manual. College of Computer & Inf. Sci., King Saud Univ., Saudi Arabia, 1988.

CR Categories and Subject Descriptors: C.0 **[General]**: Hardware/Software Interfaces; D.4.7 **[Organization and Design]**: Interactive Systems; D.4.9 **[System Programs and Utilities]**: Command and control Languages; H.4.1 **[Information Systems Applications]**: Office Automation—*equipment*; I.7.1 **[Text Processing]**: Text Editing; J.5 **[Computer Applications]**: Arts and Humanities—*linguistics*

General Terms: Design, Experimentation

Additional Key Words and Phrases: Arabic operating systems, Arabic script, automatic shape determination, bilingual typesetting, bilingual workstations, display transmutation, virtual screens, windows

ABOUT THE AUTHOR:

MURAT TAYLI is an associate professor in the Department of Information Systems at King Saud University, Riyadh, Saudi Arabia. His current research interests include distributed and real-time operating systems and the setup of their testbeds on transputers.

ABDULLAH I. AL-SALAMAH is an assistant professor in the Department of Information Systems at King Saud University, Riyadh, Saudi Arabia. His current research interests include office automation systems, programming languages, and computer arabization.

Authors' Present Address: College of Computer and Information Sciences, P.O. Box 51178, Riyadh 11543, Saudi Arabia. email for M. Tayli: f60c002@saksu00.bitnet.