Arabic Typography: A Survey

Aqil Azmi and Abeer Alsaiari

***Abstract*— For many years, tools authoring for e-documents composition has been tailored for the English script's needs. The localization of these tools to Arabic alphabet based writings is not an easy task, because of the great typographical and structural characteristic differences, and also because Arabic typography needs to process the text and analyze it contextually so that it can be typed according to the strict rules of Arabic calligraphy. Most typesetting tools seek to composite Arabic documents through adaptation of fonts capability to Latin based systems or by adaptation of the tools themselves to Arabic writing rules, and this is not an easy job due to the necessity of complex contextual analysis of Arabic writing and processing algorithms which differ from those in Latin based systems.**

The main purpose of this survey is to explore Arabic typography rules and to review the current Arabic typesetting systems. Such systems have been adapted to process special Arabic documents such as Scientific, Qur'anic and Multilingual documents. Research trends to date are summarized, and challenges confronting the development of high quality Arabic typography are identified. These challenges include the need for better algorithms for justification of Arabic text without relying on the Kashida, diacritical mark positioning and composition of dynamic fonts.

***Index Term*— Arabic, Encoding, Font, Typesetting.**

I. Introduction

The Arabic language is native to roughly three hundred million people. The Arabic script is one of the most used in the world, not only by Arabs but also by the Islamic world as it is the script used to write the Qur'an, the holy book of Muslims. Moreover, the Arabic script is used, in various slightly extended versions, to write many major languages. It is constituted in its basic form by 28 letters including 3 long vowels. Additionally there are short vowels, a total combination of 13.

The process of typesetting languages using the Arabic script is more challenging and more complex than typesetting using the Latin script because of the requirement for special needs and strict rules. The contextual rules of the Arabic script are independent of the language, font and style and have no exception.

In terms of Arabic e-document composition, the current tools still lack the ability to accommodate all of the Arabic script characteristics. Arabic script will benefit from the development of smart fonts to deal with complex script.

II. Arabic Writing Characteristics

In terms of Arabic type design, the typographer must take into account a number of characteristics and rules of Arabic script. A good awareness of these characteristics leads to professional design of Arabic type.

A. *Direction of writing*

Arabic is a unidirectional script in which the writing spreads out from right-to-left. Nowadays, Arabic mathematical documents adopt Latin alphabetic symbols which has led some to believe that Arabic writing is bidirectional because of the current mixing of Arabic strings with Latin based expressions [8].

B. *Cursivity*

The Latin based writing is based on the use of independent characters. In Arabic, only the cursive style is allowed. This cursivity implies four different forms for the same letter according to its position in the word: initial, middle, final and isolated: ج ـجـ ـج ج [2][8].

C. *Ligatures*

Arabic script is extremely rich in ligatures due to the cursive nature of writing. Some ligatures are mandatory while others are optional and exist only for aesthetic reasons, legibility or justification [2]. As shown in the following examples, the ligature can appear in various degrees:

لمحة ← لمحة ← لمحة

محمد ← محمد ← محمد

D. *Diacritic dot*

Diacritic dots are a measurement unit marked by the feather of the used calligraphy pen [5][8]. The semantic role of diacritic dots is that certain letters are characterized by the presence, number and positions of these dots [5]. For example, the basic glyph ب gives several letters according to the number of diacritic dots which appear above or below: ت, ب and ث. It is also used by calligraphers as a measurement unit to regularize the dimensions and the metrics of glyphs (Figure 1) [5][8].

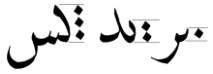
Fig. 1. Arabic letter *Alef* metrics

E. Diacritic signs

Diacritic signs (or short vowels) are markings added above or below the letters to aid in proper pronunciation of the purely consonantal text. The diacritic signs take different heights, not only with respect to basic glyphs but also according to other contextual elements. The Arabic letter can be compared to a magnet for the diacritic mark [1][8].

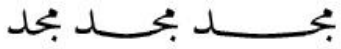
F. Allograph

Allographs are the various shapes that a letter can take according to its neighboring letters and sometimes according to the presence of Kashida. For example, the initial form of *Beh* can take three allograph shapes according to its left neighboring letter [5]:



G. Kashida

Kashida is a connection between Arabic letters and it is not a separate character but a stretch of the previous letter; used



for various purposes: Emphasis, Legibility, Aesthetic and Justification [8]. Like ligatures, Kashida comes in various degrees:

III. Arabic Type Evolution

The first printing agency was founded in Cairo in the 19th century. In that period, Arabic was typed without using a keyboard; instead four or more cases with metal glyphs were used (see Figure 2) [10][15]. The total amount of glyphs can reach to about 500 including most calligraphy ligatures, vowels, Qur'anic punctuation and allographs of letters. Typing with such cases and selection of contextual allograph of letter had no specific mechanism; it was completely dependent on the skill of the typesetter and his familiarity with calligraphic practice. In 20th century, a standard typecase took place in Cairo to match the quality of well-written manuscripts, known as *Almatba'a Al'amiria* (المطبعة الأميرية). This typecase is divided into four parts and uses a total of 470 characters and has been kept in use until today; books typeset in a traditional way, throughout the Arabic world, are still using the same set of characters, and the same conventions and rules [10].

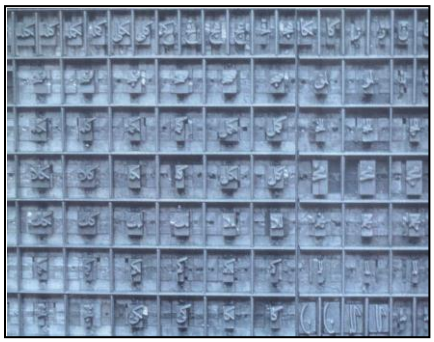


Fig. 2. Movable typecase

In 1945, the Cairo Academy for the Arabic Language launched a worldwide competition to simplify Arabic because of the need for a new simplified script to fit with the new type techniques (i.e. the typewriter). Many proposals were submitted to the Academy but none of them was accepted.

The Academy elaborated its own project of typographical simplification of Arabic. It aimed to reduce the number of types and use only available glyphs, proposing to use about 72 types as shown in Figure 3.

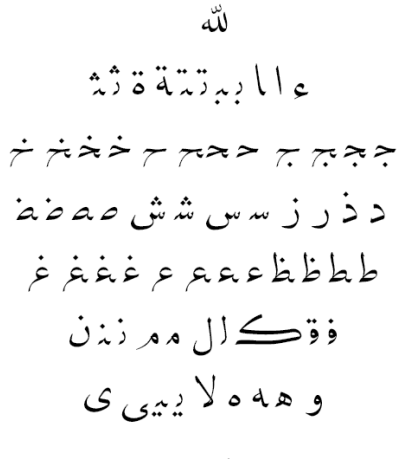


Fig. 3. The Academy types

Most of the types of the Academy typeface, either exist already in the traditional one, or can been obtained by cutting off parts of types of the latter except the ain and ghain letters for which one has to draw new types [11].

During the 60's until the late 80's, conversion from mechanical typesetting to computer typography was not an easy task. Due to technical limitations the computer was not able to cope with the complexity of advanced type cases such as *Almatba'a Al'amiria*. Therefore, the computer industry has tried to impose new standards of simplified typesetting, covering most of the time only the fundamental properties of Arabic script.

In the 80's, the company DecoType invented the Arabic Calligraphic Engine (ACE) to deal with complex Arabic [18]. DecoType's ACE technology arose from the concept of smart font technology, paving the way for what was to become OpenType.

After the mid 90's till our present day, the invention of smart font technology like OpenType Font supports Arabic type on all major computer platforms and facilitates the composition of complex scripts. Smart font capabilities allow composition of ligatures but on the other hand, it still cannot handle some of the ligatures in a proper way.

IV. Arabic Type Design

The characters used in text are: the letters of the alphabet, numbers and symbols. A font is a homogeneous set of characters. Regardless of its form (initial, middle, final, isolated), each letter has a single code to represent it within the computer. The system uses contextual analysis algorithms to determine the form of letters based on their position in a word.

All forms of all the letters are stored in computer memory. Many encoding systems exist to represent a specific language or a specific set of characters. It is noted that different encoding systems may conflict with each other since the same code may be assigned to two different characters in two different systems. These led to the invention of the standard encoding system "Unicode".

A. *Unicode*

Unicode is a world-wide character encoding standard offering a special code for each symbol used by the computer regardless of the operating system, software, and the used language. It encodes all the symbols of the written languages in the world [3].

The Unicode standard distinguishes between *characters* and *glyphs*. It encodes only the default character set no matter how many contextual representations (glyphs) it may exhibit in text. Text rendering engines take the character code from the keyboard and transfer it from code space into glyph space to determine the appropriate corresponding glyph [6]. For example, when the following letters are entered by keyboard:

U+062F	U+062C	U+0645
د	ج	م

The program uses these code numbers as indices to determine the corresponding glyph based on its position in the word:

مجد ⟵ مـ جـ د

In additional to a unique code, Unicode offers a specific name for each character. For example, the letter *Alif* has a code 0627 and a name 'ARABIC LETTER ALEF'. The range of Unicode chart for Arabic starts from U+0600 to U+06FF representing the standard set of characters used in Arabic-based scripts [1][17]. The overall characters are divided into the following charts:

- Arabic (0600–06FF): encodes the standard set of Arabic characters.
- Arabic Supplement (0750–077F): encodes contextual forms mostly used for writing African languages.
- Arabic Presentation Forms-A (FB50–FDFF): encodes contextual forms and ligatures used in Persian, Urdu, Sindhi and Asian languages.
- Arabic Presentation Forms-B (FE70–FEFF): encodes spacing forms of Arabic diacritics, and more contextual forms.

B. *Font format*

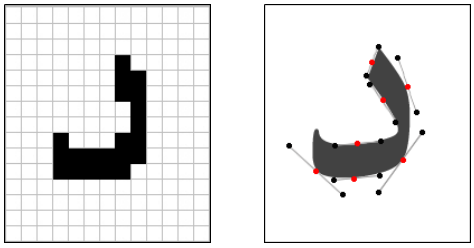
Font is a data file used to store, represent and reproduce a collection of characters in a form suitable for digital output devices. The coding of its glyphs follows some character encoding system based on the current software environment. Most modern font formats are supported by the Unicode encoding system. Font can appear in slightly different designs like regular, *italic*, **bold** and ***bold italic***; all together they constitute the font family. Technologically, there are two basic kinds of computer font file data formats, differing mostly in their representation format, efficiency and capability to scale up [12]:

- *Bitmap fonts*

Glyph is represented by a matrix of pixels in each face and size and simply specifies which pixels must be turned on (see Figure 4a). Therefore, these fonts are not scalable and require a complete set of glyphs for each size. Compared to other types, bitmap fonts are much faster and easier to use. They were suitable for early computer systems due to their technical limitations.

- *Outline fonts*

Glyph is defined by a set of lines and curves that form the glyph outline (Figure 4b). The mathematical formulation of glyphs makes the font scalable to any size without any loss of quality. Unlike bitmap fonts, variation designs such as *italic*, **bold** and ***bold italic*** can be created easily from a given font. To be displayed on screen or in print requires rendering the outline into a bitmap which means more complicated work. Under this technology a number of font formats were developed; such as PostScript font format (also known as Type 1) which was developed by Adobe. Then, Apple developed its own technology called TrueType font format, and traded this technology with Microsoft. The OpenType font's format, developed jointly by Adobe and Microsoft, combines the advantages of TrueType and Type 1 formats together and adds several new features. METAFONT is a font design program written by D.E. Knuth to create suitable outline fonts for its software TeX.



(a) (b)

Fig. 4. Font formats: (a) bitmap; (b) outline

C. *Outline fonts*

In early font technologies, only one-to-one mapping existed between characters and glyphs, a feature completely accommodated to Latin scripts. In Arabic script this issue is more complex, because it differs in many characteristics, making it incompatible with this technique. Arabic script requires a reordering before rendering the text, and also the variation of character forms according to the position in the word requires the existence of glyph substitution and positioning features in font technology [9].

More intelligent fonts have been developed like OpenType format to support complex scripts, and these sustain by themselves the process of contextual analysis and other typographical procedures.

These fonts have a clear distinction between the letter and the form it is represented by, and all other rules that control complex scripts are figured out by tables.

- *OpenType*

One of the most practical smart fonts developed to support complex scripts and enhance a font's typography. The font character encoding is based on Unicode to provide support for lots of different languages where individual OpenType fonts can have up to 65,000 glyphs [7]. OpenType tables include the basic tables of TrueType format, such as '*cmap*' which maps the font character to its Unicode number. Also, extended tables were added to support complex scripts [3][4]:

- Glyph Substitution Table (GSUB)

Includes the information that is required to substitute the character through a number of feature tags:

- *GSUB Lookup Type 1*

Contains a number of tags used to map default alphabetic forms to corresponding forms. For example, tag '*fina*' maps default alphabetic forms to its corresponding ending forms.

Backing store	'fina' form
ب	بـ

- *GSUB Lookup Type 3*

Replaces default character forms with corresponding swash forms based upon the context of surrounding characters.

Glyph shapes	'cswh'
من محمد	من محمد

- *GSUB Lookup Type 4*

Some sequences of glyphs constitute one ligature with each other. The tag '*rlig*' maps sequences of glyphs to their corresponding ligatures.

Glyph shapes	'rlig' form
ل + ا	لا

- Glyph Positioning Tables (GPOS)

Help to control exact character positioning in relation to each other and to diacritics:

- *GPOS Lookup Type 2*

The tag '*kern*' adjusts the space between two glyphs to make them more visually appealing.

Without kerning	د ر س
With kerning	درس

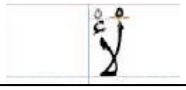
- *GPOS Lookup Type 3*

Tag '*curs*' identifies the positions of cursive characters so that the exit point of the current character matches with the entry point of the following character.

Glyph shapes	'curs'
ش + م	شم

- *GPOS Lookup Type 4*

Tag *'mark'* identifies the positions of marks in relation to glyphs or ligatures.

Mark to Mark positioning	Mark to glyph positioning
	

V. Arabic Text Processing

Although font files are platform independent, OpenType only supports complex scripts if the operating system knows how to handle them. Windows systems introduce a system component called Unicode script processor 'Uniscribe' which consists of dynamic link libraries DLL, each one of which is responsible for providing knowledge about issues on how to handle a special language. Pango engine provides the same capabilities of Uniscribe in Linux systems. MacOs is slightly different with all expertise being encapsulated in AAT font which means that AAT can handle any arbitrary language [3] [9].

In TeX, a typesetting system developed by D.E. Knuth, this issue is slightly different. TEX is a programming language, but rudimentary and somewhat flexible; it offers a large set of mechanisms that describe the appearance of the document. It takes plain text file as input and transforms it into an output representation. Both TeX and its extension Latex have been adapted to handle passages of Arabic script. ArabTeX by Klaus Lagally [13] is designed to produce Arabic script with vowel marks and most usual ligatures. TeX handles a specific language through a transliteration table which represents the phonetic spelling of that language's text in Latin alphabet, to be processed in the computer [13].

Uniscribe in windows starts processing text by dividing it into ranges of similar characteristics (language, font …). Then, it reorders the characters according to the direction of the current language.

A. *Contextual analysis*

Due to the cursive nature of Arabic script, the forms of a character vary according to its position in the word. The determination of an appropriate character glyph according to its context is accomplished through contextual analysis algorithms [16][17].

In Unicode algorithm, Arabic characters are classified into six main joining classes: Right-joining, Left-joining, Dual-joining, Join-causing, Non-joining and Transparent, and also two derived classes: Right join-causing and Left join-causing. An appropriate glyph is determined on the basis of its joining

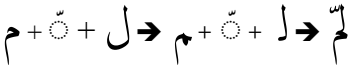
class and the joining class of adjacent characters. Each character belongs to one of the joining classes. Unicode defines seven joining rules using these classes to describe the joining behavior of characters. In the following rules, X refers to a character and its various glyphs are presented in Table I.

TABLE I
ARABIC GLYPH TYPES

Glyphs	Description
X.n	Refers to Nominal glyph form (isolated).
X.r	Refers to Right-joining glyph form.
X.l	Refers to Left-joining glyph form.
X.m	Refers to Dual-joining glyph form.
X.c	Refers to characters that do not change shape themselves but cause joining (ZERO WIDTH JOINER and TATWEEL)
X.t	Refers to characters that don't join or cause joining (like Arabic *diacritic signs*, HAMZA BELOW...).

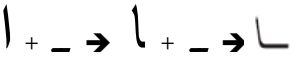
Rule 1: Transparent

X + X.t + X ➔ X.r + X.t + X.l



Rule 2: Right-joining

X.n + X.c ➔ X.r + X.c



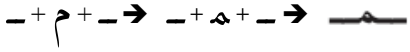
Rule 3: Left-joining

X.c + X.n ➔ X.c + X.l

There are no members in this class.

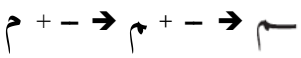
Rule 4: Dual-joining

X.c + X.n + X.c ➔ X.c + X.m + X.c



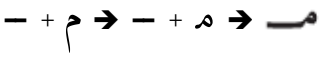
Rule 5: Dual-joining

X.n + X.c ➔ X.r + X.c



Rule 6: Dual-joining

X.c + X.n ➔ X.c + X.l



Rule 7: applies if none of the previous rules are applicable.

X ➔ X.n

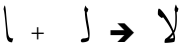
After applying the joining rules, the ligature rules are applied to describe ligature behavior.

Rule 1:

ALEF.r + LAM.m ➔ (LAM-ALEF).r

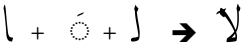
Rule 2:

ALEF.r + LAM.l ➔ (LAM-ALEF).n



Rule 3:

ALEF.r + X.t + LAM.l ➔ (LAM-ALEF).n + X.t



B. Justification

Justification is widely used in Arabic calligraphy and may good mechanisms to make paragraphs appear visually homogeneous have been developed by calligraphers.

Justification in Latin typography is applied by handling a typographical hyphenation and inter-word spaces. Hyphenation is the breaking of words when they come at the end of a line at a determined point, where the given word supports breakage. Inter-word spaces give the line flexibility by inserting variable sized spaces called glue, which can be stretched or shrunk. Unequal and large glues sometimes break up the visual structure of a text. Hàn Thé Thành [3][5] proposed a solution by slightly modifying the width of characters.

In Arabic text justification, hyphenation is not allowed and so it is a more involved task than Latin script. Justification mechanisms are derived from Arabic calligraphers' skills and Arabic script features:

- *Kashida*

Kashida is the extension of a certain letters to make words wider in order to produce paragraph alignment. In Arabic calligraphy, the placement of Kashida is based on a complex set of rules giving priority to some letters over others and the selection of characters to be stretched is called *tansil*.

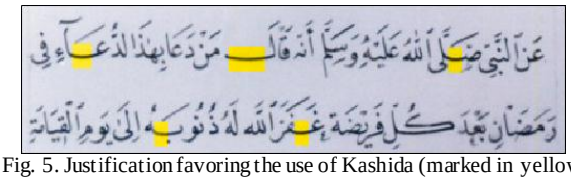


Fig. 5. Justification favoring the use of Kashida (marked in yellow)

In digital typography, the composition of such variable sizes of curvilinear Kashida is one of the most complex tasks. Therefore, and due to technical limitations, most Arabic typesetting software insert straight line segments between letters in terms of justification. One solution consists of inserting predefined glyphs with different sizes, which are elements of a font. A better approach consists of building a so called 'dynamic font' which parameterizes the composition procedure of Kashida and computes Bezier curve dynamically. The parameters determine the level of extensibility of Kashida [3]. The composition of Kashida is only one aspect of a more general problem in the composition of dynamic font.

The CurExt system was developed to composite both

vertically and horizontally extensible mathematical symbols in curvilinear fashion [14].

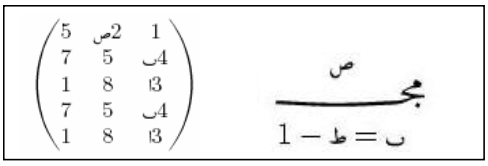


Fig. 6. Stretched mathematical symbols

Many letters can be permutated with a number of alternatives shapes. This methodology was first used by Gutenberg who systematically applied ligatures and permutated some letter shapes with different ones to optimize justification on the line level.

OpenType format employs this feature in terms of justification through the use of justification table "jalt". The table "jalt" offers the ability to provide alternate width glyphs that can be substituted upon available width. The alternatives are organized as priority levels. First, the substitution table "Lookup features" is applied in its normal way to determine the appropriate glyph for the default character. Next, the rendering engine divides the paragraph into lines and computes the badness value for each line separately. If any line receives a badness value, the rendering engine applies the level 1 Lookup by either decomposing/composing ligatures or substituting glyphs with alternatives to get wider/narrower words. If the line obtains a new badness value, it applies the level 2 Lookup and so on [3][4].

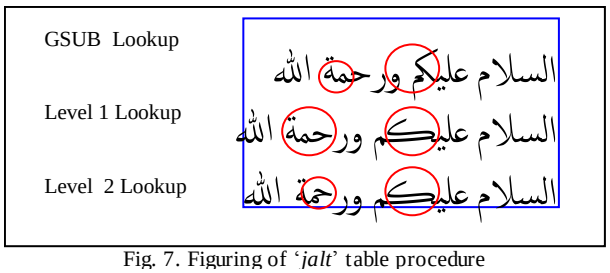


Fig. 7. Figuring of '*jalt*' table procedure

This feature to improve justification minimizes the requirement for using Kashida. Indeed, it must be implemented through an application but there is no application that currently employs this feature.

VI. Conclusion

With the spread of personal computers and processing tools, typography has become the responsibility of the author. Arabic alphabetic script requires that additional care be taken to leave its graphical structure intact. The development of document composition tools has facilitated the task of typography. The latest research trends present contributions to solving the problems related to Arabic typography but much more is still required as the field is still far from meeting all the requirements of Arabic calligraphy rules. The current typesetting systems based on Kashida do not take into account their positioning rules. According to calligraphic rules, certain letters can be stretched while others can be compressed. This requires the development of powerful algorithms that formalize Arabic calligraphy rules to achieve a high quality output acceptable from the Arabic calligraphic point of view.

References

[1] محمد حسيني، عزالدين لزرق، ومحمد جمال الدين بن عطية. "علامات الشكل في المستند الإلكتروني العربي". المؤتمر الدولي الرابع لممارسات علوم الحاسب باللغة العربية. جامعة قطر، الدوحة، قطر.(1-4 ابريل 2008).

[2] عزالدين لزرق، "أبناط في معالجة النصوص العلمية العربية". ندوة تقنية المعلومات والعلوم الشرعية والعربية. جامعة الإمام محمد بن سعود الإسلامية، الرياض، السعودية. (6-7 مارس 2007).

[3] محمد اليعقوبي، عزالدين لزرق، ومصطفى بنوني."نحو تيبوغرافية ديناميكية لمحاذاة النص العربي". المؤتمر الدولي الرابع لممارسات علوم الحاسب باللغة العربية. جامعة قطر، الدوحة، قطر.(1-4 ابريل 2008).

[4] http://www.microsoft.com/typography

[5] M.J.E. Benatia, M. Elyaakoubi, and A. Lazrek, "Arabic text justification", *Proc. 2006 Annual Meeting, TUGboat*, 27 (2), 2006, pp. 137-46.

[6] C. Bigelow and K. Holmes, "The design of a Unicode font", *Electronic Publishing*, 6 (3), 1993, pp. 289-305.

[7] M. Eastman, "OpenType Fonts: The Next Level of Digital Typography in a True Cross-platform Font Format", *Communication Arts Photography Annual*, Aug. 2002, pp. 208-11.

[8] M. Elyaakoubi and A. Lazrek, "Arabic scientific e-document typography", *Proc. 5th Int. Conf. on Human System Learning (ICHSL5)*, Marrakech, 22-25 Nov. 2005, pp. 241-52.

[9] C. Fynn, "'Smart' Font and Shaping Systems for Complex Asian Script", *The Tibetan & Himalayan Digital Library*, Jan. 2004. Available: http://www.thdl.org/xml/showEssay.php?xml=/tools/fonts/smartfonts.xml

[10] Y. Haralambous, "The Traditional Arabic Typecase, Unicode, TEX and METAFONT", *TUGboat*, 18 (1), 1997, pp. 17-23.

[11] Y. Haralambous, "Simplification of the Arabic Script: Three Different Approaches and their Implementations", in *LNCS* 1375, Springer-Verlag Berlin/Heidelberg, 1998, pp. 138-56.

[12] V. Kurz, "Methods for character description Overview of font formats", 17 Nov. 2000.

[13] K. Lagally, "ArabTEX, a System for Typesetting Arabic", *Proc. 3rd Int. Conf. and Exhibition on Multi-lingual Computing: Arabic and Roman Script (ICEMCO92)*, Univ. Durham, UK, 10-12 Dec. 1992, pp. 9.4.1-9.4.8.

[14] A. Lazrek, "CurExt, Typesetting variable-sized curved symbols," *Proc. EuroTeX 2003: 14th European TeX Conf., TUGboat*, 24 (3), 2003, pp. 323-27.

[15] T. Milo, "ALI-BABA and the 4.0 Unicode Characters – Towards the Ideal Arabic Working Environment, New input output concepts under Unicode," *Proc. EuroTeX 2003: 14th European TeX Conf., TUGboat*, 24 (3), 2003, pp. 502-11.

[16] K.F. Moshfeghi, "A New Algorithm for Contextual Analysis of Farsi Characters and Its Implementation in Java", *17th Int. Unicode Conference*, San Jose, CA, Sep. 2000.

[17] The Unicode Consortium, *The Unicode Standard, Version 5.0*, Boston, MA: Addison-Wesley, 2007.

[18] P. Zoghbi, "History of Arabic Type Evolution from the 1930's till present", May 2007. Available: http://29letters.wordpress.com/2007/05/28/arabic-type-history.