The United Nations
University

UNU/IIST

International Institute for
Software Technology

Multi-directional Multi-lingual Script Processing

Myatav Erdenechimeg and Richard Moore

June 1996

UNU/IIST Report No. 75

$\boxed{\mathcal{T}}$

UNU/IIST

UNU/IIST enables developing countries to attain self-reliance in software technology by: (i) their own development of high integrity computing systems, (ii) highest level post-graduate university teaching, (iii) international level research, and, through the above, (iv) use of as sophisticated software as reasonable.

UNU/IIST contributes through: (a) advanced, joint industry–university advanced development projects in which rigorous techniques supported by semantics-based tools are applied in case studies to software systems development, (b) own and joint university and academy institute research in which new techniques for *(1) application domain* and computing platform modelling, *(2) requirements capture,* and *(3) software design & programming* are being investigated, (c) advanced, post-graduate and post-doctoral level courses which typically teach Design Calculi oriented software development techniques, (d) events [panels, task forces, workshops and symposia], and (e) dissemination.

Application-wise, the advanced development projects presently focus on software to support large-scale infrastructure systems such as transport systems (railways, airlines, air traffic, etc.), manufacturing industries, public administration, telecommunications, etc., and are thus aligned with UN and International Aid System concerns. UNU/IIST is a leading software technology centre in the area of infrastructure software development.

UNU/IIST is also a leading research centre in the area of Duration Calculi, i.e. techniques applicable to *real-time, reactive, hybrid & safety critical systems.* The research projects parallel and support the advanced development projects.

At present, the technical focus of UNU/IIST in all of the above is on applying, teaching, researching, and disseminating Design Calculi oriented techniques and tools for trustworthy software development. UNU/IIST currently emphasises techniques that permit proper development steps and interfaces. UNU/IIST also endeavours to promulgate sound project and product management principles.

UNU/IIST's primary dissemination strategy is to act as a clearing house for reports from research and technology centres in industrial countries to industries and academic institutions in developing countries. At present more than 200 institutions worldwide contribute to UNU/IIST's report collection while UNU/IIST at the same time subscribes to more than 125 international scientific and technical journals. Information on reports received (and produced) and on journal articles is to be disseminated regularly to developing country centres — which are then free to order a reasonable number of report and article copies from UNU/IIST.

Dines Bjørner, Director — 2.7.1992–1.7.1997

UNU/IIST Reports are either $\mathcal{R}$esearch, $\mathcal{T}$echnical, $\mathcal{C}$ompendia or $\mathcal{A}$dministrative reports:

$\boxed{\mathcal{R}}$ Research Report • $\boxed{\mathcal{T}}$ Technical Report • $\boxed{\mathcal{C}}$ Compendium • $\boxed{\mathcal{A}}$ Administrative Report

The United Nations
University

UNU/IIST

International Institute for
Software Technology

P.O. Box 3058
Macau

Multi-directional Multi-lingual Script Processing

Myatav Erdenechimeg and Richard Moore

Abstract

Although there are increasingly many text processing systems which support multiple languages, the majority of these do not allow the different languages to retain their traditional writing direction, instead imposing either that of the European families of languages (each line of text read left to right; lines ordered top to bottom) or that of the language of the country in which the software was written. In this paper we present a model of documents which not only allows the writing direction of a document as a whole to be defined but also describes multi-lingual documents in which different pieces of text have different writing directions. We also discuss the design of a software system based on this model which allows text in different languages which has been generated using existing text processing tools to be combined into a single multi-lingual document.

Myatav Erdenechimeg is working as a UN Fellow at UNU/IIST from September 1995 until August 1996. She graduated from the National University of Mongolia with a degree in Mathematical Calculation and is a lecturer in computer science at the National University of Mongolia. Her more recent work has concentrated on the problem of the computerisation of the Mongolian language. As part of this she has developed coding systems for both Mongolian script and Mongolian Cyrillic letters and software for Mongolian word processing. She has also worked on software for translation between Mongolian script and Cyrillic. She has published several papers in these fields.

Richard Moore is a Research Fellow on the staff of UNU/IIST, a position he took up on October 1st 1995. He has an M.A. in mathematics from the University of Cambridge and a Ph.D. in physics from the University of Manchester. He has been engaged in computing science research in the field of formal methods since 1985, a large part of which was carried out in the formal methods group at Manchester University, and he is co-author of two books on formal methods – mural: a Formal Development Support System; and Proof in VDM: A Practitioner's Guide. He has also worked for the Defence Research Agency in Malvern, UK, on various formal methods projects both as a consultant and as a full-time member of staff.

Copyright © 1997 by UNU/IIST, Myatav Erdenechimeg and Richard Moore

Contents

1 Introduction **1**

2 SUDAR – Software Support for the Mongolian Language **2**

3 An Analysis of Multi-lingual Documents **4**

4 A Formal Model of Multi-lingual Documents **9**

5 Software Support for Multi-Directional Multi-Lingual Texts **13**

5.1 Creating and editing documents 14

5.2 Defining the physical layout of documents 14

6 Conclusions **16**

1 Introduction

There are several thousand languages in use in the world today, for an increasing number of which some software support is available. For many of the European families of languages, such support is relatively easy to add on to existing Roman[1] text and document processing systems, often requiring little more than the addition of a few extra characters to the font and the assignment of these characters to keys on the keyboard. For other languages, notably the oriental and Arabic families, things are much more difficult for several reasons.

First, these languages tend to have their own character sets which are not simple extensions or modifications of the English character set. This means that existing fonts generally cannot be used and whole new fonts have to be designed to support these languages.

Second, many of these languages have character sets which are much larger than those of the European languages, sometimes incorporating thousands of distinct characters. For such languages there are clearly not sufficient keys on the keyboard to allow a simple mapping from a single key to a single character as is done on an English keyboard. Instead more complicated input methods are required, often involving the use of multiple keystrokes or some sort of graphical interface to generate a single character.

Finally, many of these languages are not traditionally written in the same direction as the European languages, some, particularly those of the Arabic family, being written horizontally but right to left instead of left to right, others being written vertically. In fact this latter group is itself subdivided into languages in which the lines[2] of characters are ordered right to left (for example Chinese, Japanese and Korean) and those in which they are ordered left to right (for example Mongolian).

Where the software support for these languages has been provided by extending some existing Roman-based software, the extension has tended to be concentrated on the first two aspects, namely on the provision of the extra character sets (fonts) and the modification or extension of the keyboard input mechanism, and the question of directionality has mainly been ignored. To a large extent this is of course to be expected, as without the ability to display the characters of a language or to input them into the computer in some way the question of the reading and writing direction is largely academic. As a consequence, however, people wishing to use these systems to process non-Roman texts, particularly those of the oriental languages, have generally been forced to do so using the standard European directionality. This is unnatural and can make reading and writing the text extremely difficult[3].

[1] Here 'Roman' means any language whose character set consists of the basic English alphabet, possibly modified by the addition of accents to some letters or extended with such accented letters.

[2] Throughout this paper the word 'line' in the sense of 'lines of text' is used generically to refer to both horizontal and vertical text and should not be taken to imply that the text is horizontal. The word 'column' is also used to describe vertical text.

[3] deedni ro) sdrawkcab hsilgnE etirw ot ro sdrawkcab nettirw hsilgnE daer ot rehtie deirt sah ohw enoyna sA
!etaicerppa ylidaer dluohs (noitcerid dradnats-non rehto yna ni

Software support which was primarily designed to support non-Roman languages often does allow its users to write and read documents in the appropriate "natural" direction and sometimes allows the user to choose the reading and writing direction of a document. However, even such systems are not truly multi-lingual as they do not allow this direction to change within the document.

As an example of this, we describe a text processing system which supports the various forms of the Mongolian language (Mongolian script, and Cyrillic and English transliterations) in Section 2. We point out its limitations and, based on these and on an extensive study of multi-lingual documents which we summarise in Section 3, we go on to propose a generic model of multi-lingual documents which not only allows the overall reading and writing direction of the whole document to be defined but which also allows different parts of the document to have different reading and writing directions. In the final section of the paper we discuss some requirements for a software system supporting this model.

2 SUDAR – Software Support for the Mongolian Language

The Mongolian language exists in three different forms. The traditional form is Mongolian script, which was in common use in Mongolia up to 1946 and which became its official language once more in 1994. Between 1946 and 1994 a Cyrillic transliteration of this was used as the official language of the Mongolian People's Republic, and a transliteration of this to the English alphabet also exists which is commonly used in the field of computing in particular.

Mongolian script is properly written vertically in columns, these being ordered from left to right. Words are separated by spaces, but individual letters within a word are joined together (see Figure 1) and the displayed form of each letter may depend both on the preceding letter, with which it can form a ligature (see Figure 2), and on its position in the word, specifically on whether it appears at the beginning (initial form), in the middle (medial form) or at the end (final form) of the word. In some cases there are even several different medial and final forms of the same letter (see Figure 3).

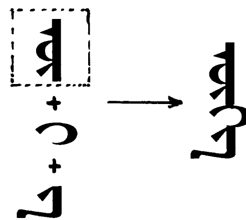


Figure 1: Joining of Characters in Mongolian Script

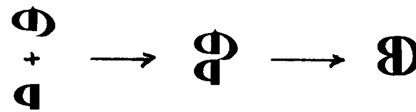


Figure 2: Mongolian Script Ligatures

	initial	medial	final
a - letter			
ö - letter			
b - letter			

Figure 3: Initial, Medial and Final Forms of Mongolian Script Letters

Both the Cyrillic and the English transliterations are written horizontally in the standard Roman style, though the Cyrillic form uses an alphabet which has two more characters than that used for the Russian language.

A word processing system for Mongolian, which runs on a PC and is called SUDAR, has been developed at the National University of Mongolia by M. Erdenechimeg, U. Namsraijab, S. Lodoysamba and D. Dashcheden. SUDAR supports the Mongolian script as well as the Cyrillic and English transliterations, and also allows these to be intermixed in the same document. For the Mongolian script, it automatically generates the correct positional form (initial, medial or final) of each letter and the correct ligature with the preceding letter if appropriate, so the user of the system does not have to worry about inputting the correct letter form.

SUDAR also supports both horizontal and vertical reading and writing directions. Figures 4 and 5 show the same piece of text presented in these two different styles, and SUDAR allows the user to change between these two styles very easily for any document. However it does not allow the reading and writing direction to change within a document, so that in documents such as those in the figures where Mongolian script is mixed with the transliteration forms only one of these can be presented with its correct orientation and the other appears not only in an unnatural orientation but also with the characters rotated through 90° (though as we point out in Section 3 the rotation of characters in multi-lingual documents is by no means uncommon).

Although SUDAR fulfils the needs of many users who wish to construct documents in Mongolian, those needing to produce (multi-lingual) documents in which the three different forms of the

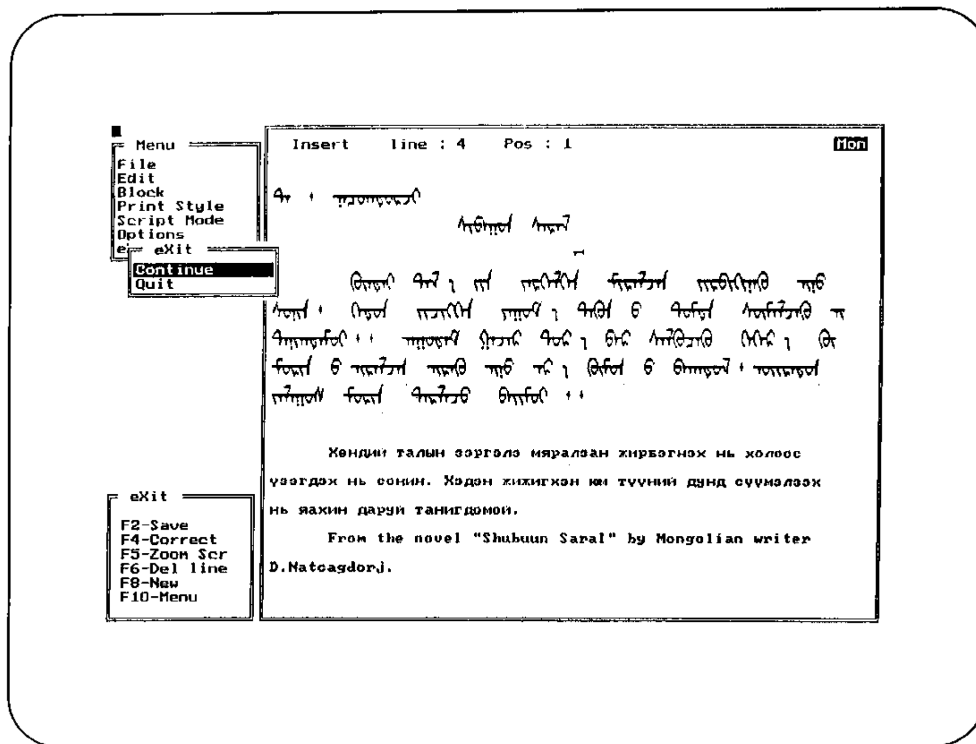


Figure 4: SUDAR: Horizontal Mode

Mongolian language are intermixed in such a way that each retains its natural reading and writing direction clearly require tool support which is more powerful. However, a document which is only multi-lingual in the sense of mixing the different forms of the Mongolian language has much in common with multi-lingual documents in general. Indeed, if we can design software which supports the production of general multi-lingual documents than it will also support the production of documents in which the different forms of Mongolian are intermixed and retain their correct orientation. We therefore discuss general multi-lingual documents in the next section.

3 An Analysis of Multi-lingual Documents

An extensive study of a wide range of different multi-lingual documents reveals that there are essentially two different ways in which texts in different languages, possibly with different reading and writing directions, can be intermixed in a single document. In the first of these, the parts of the text with the different languages and/or orientations are clearly separated, as in the Mongolian examples shown in Figures 4 and 5 and in the example shown in Figure 6 in which

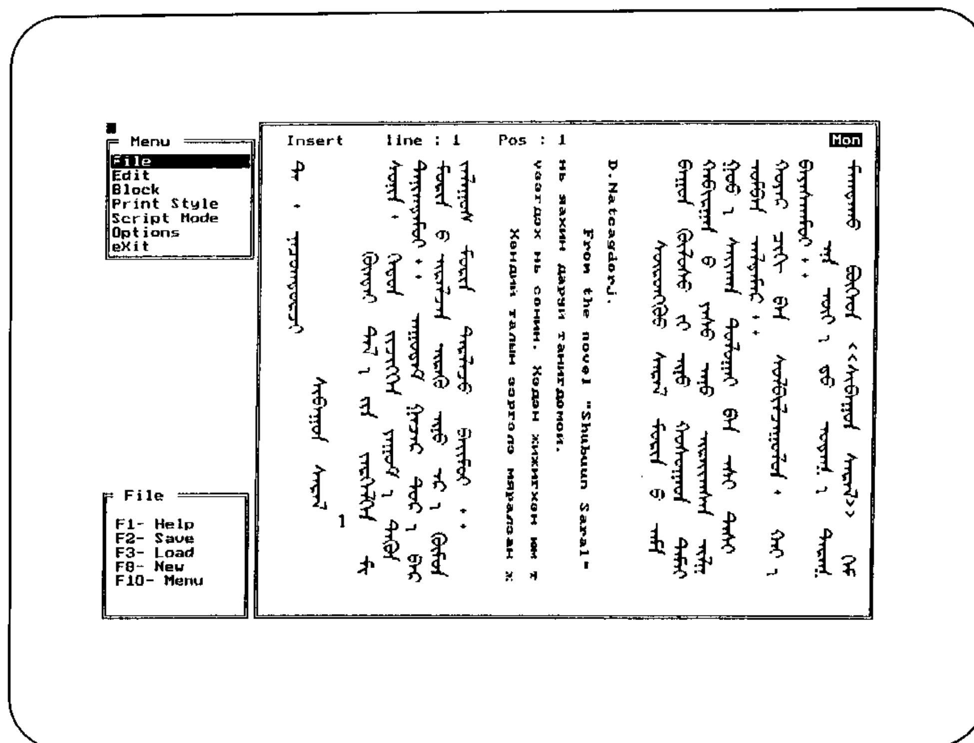


Figure 5: SUDAR: Vertical Mode

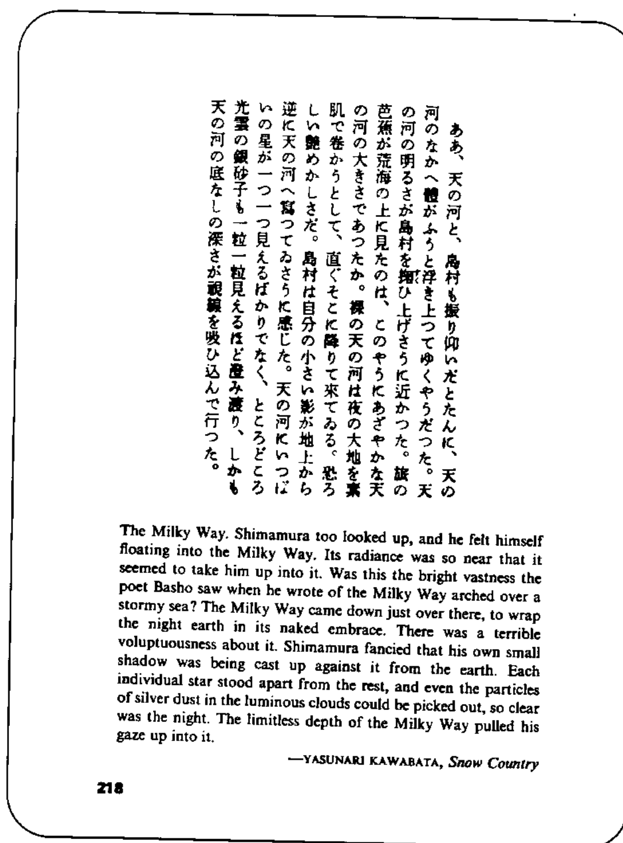
a Japanese text (written vertically in columns and beginning at the top right-hand corner) is accompanied by its English translation.

In the second type, parts of one language are embedded in the other, so that, for example, a sentence in one language is "interrupted" by insertions in another language. If both languages have the same natural orientation this is used by both, but there are again two different basic ways in which the insertion can happen in cases where the two languages have different natural orientations.

First, the embedded language can retain its own natural orientation as in the example shown in Figure 7, in which German text is interrupted by insertions in Mongolian script [2], and in the middle example of Figure 8, in which English text is interrupted by insertions in Arabic, the Arabic retaining its traditional horizontal and right-to-left orientation[4].

Alternatively, the embedded language loses its natural orientation and instead adopts that of the language in which it is embedded. In this case the embedded language is often written with

[4]Note how this causes the two Arabic words to appear in a different order when the English word 'and' which separates them in the first sentence is removed.



ああ、天の河と、島村も振り仰いだとたんに、天の河のなかへ體がふうと浮き上つてゆくやうだつた。天の河の明るさが島村を掬ひ上げさうに近かつた。旅の芭蕉が荒海の上に見たのは、このやうにあざやかな天の河の大きさであつたか。裸の天の河は夜の大地を素肌で巻かうとして、直ぐそこに降りて來てゐる。恐ろしい艶めかしさだ。島村は自分の小さい影が地上から逆に天の河へ寫つてゐさうに感じた。天の河にいつぱいの星が一つ一つ見えるばかりでなく、ところどころの光雲の銀砂子も一粒一粒見えるほど澄み渡り、しかも天の河の底なしの深さが視線を吸ひ込んで行つた。

The Milky Way. Shimamura too looked up, and he felt himself floating into the Milky Way. Its radiance was so near that it seemed to take him up into it. Was this the bright vastness the poet Basho saw when he wrote of the Milky Way arched over a stormy sea? The Milky Way came down just over there, to wrap the night earth in its naked embrace. There was a terrible voluptuousness about it. Shimamura fancied that his own small shadow was being cast up against it from the earth. Each individual star stood apart from the rest, and even the particles of silver dust in the luminous clouds could be picked out, so clear was the night. The limitless depth of the Milky Way pulled his gaze up into it.

—YASUNARI KAWABATA, *Snow Country*

218

Figure 6: Japanese Text with English Translation

its characters rotated when one of the natural orientations is horizontal and the other vertical. This is illustrated in the first and third examples in Figure 8. The first of these shows English text in which Mongolian script is inserted. The characters of the Mongolian script are rotated through 90° anticlockwise. The other example shows Chinese text written in its traditional orientation (vertically top to bottom with the columns progressing from right to left) in which English text is embedded. In this case the characters of the English text are rotated through 90° clockwise.

We define a *frame* to be a piece of text written in a single language[5] and within which both the reading and writing direction and the rotation of the characters do not change. In more physical terms a frame can also be thought of as a rectangular area (on a computer screen or on a piece

[5]More strictly perhaps, which can be generated using the characters of a single font. We can clearly produce, say, German text with English insertions using only a German font (and keyboard): there is effectively nothing which distinguishes the two languages at this level because the English character set is a subset of the German.

Syntax. Bildung der Sätze. 129

wenn du gleich nicht zu mir kommst (kämest), *so komme ich doch zu dir;* oder auch *wenn du gleich nicht zu mir gekommen wärest, so wäre ich doch zu dir gekommen*, oder *würde zu dir gekommen seyn.* Ferner mit dem Accusativ des Infinitivs und mit Umwandlung des persönlichen Fürworts in das Possessivum *ich habe es längst gewusst, dass du nicht zu mir kommen würdest* (wörtlich: *dein zu mir Nichtkommen ich längst gewusst habe*).

§. 201. Statt fernerer Satzproben lasse ich zur Uebung das zweite Capitel des *Üligerün Dalai* (Meer der Gleichnisse), nebst den nöthigen Erläuterungen der Wort- und Satzformen und der Uebersetzung hier folgen. Es ist dasselbe, als Erzählung und des leichten Styles wegen, vorzüglich geeignet, den Bau der Mongolischen Sprache kennen zu lernen.

Leseübungen.

Figure 7: German Text with Mongolian Script Insertions

of paper) within which the text is displayed or printed.

The orientation of a frame is determined by its *entry point* and its *line stream.* The entry point defines the position at which reading or writing begins, and is situated either at the top left-hand corner or the top-right-hand corner of the frame. We are not aware of any languages in which reading and writing start at the bottom of the frame, so it is sufficient to define the entry point simply as either left or right. The definition can very easily be extended to incorporate the lower corners should this prove necessary.

The line stream defines the direction in which the text is read or written and is either horizontal

For example you can see on the chart the forms of a letter
"a" at the begining (code is 64); in the middle (96)
(common), (128 (in the case of); at the end it has 3
different forms (160), (192) (224) (in the case
of)

(a)

The words الإسلام and العرب mean Islam and the Arabs.
The words الإسلام و العرب mean Islam and the Arabs.

(b)

與記憶同一陣線的不只空間，還有時間。
魏紹恩說，「王家衛對時間執着，

優質混沌

並且態度堅決。」
的確如此。
王家衛電影的英文名字最了當，無論直接或間接，幾乎全都牽涉時間。
《旺角卡門》——《As Tears Go By》
《阿飛正傳》——《Day of Being Wild》
《重慶森林》——《Chungking Express》
《東邪西毒》——《Ashes of Time》
對時間的執迷，成了王家衛電影最不可或缺的元素之一。
《阿飛正傳》，張國榮便要張曼玉永遠準確無誤地記住一九六零年四月十日下午三時前的一分鐘。

長舌黃曆

梯 亞

同樣是關於時間，金城武在《重慶森林》裡的失戀故事很好玩。女朋友阿 May 拒絕見面，金城武便私底下定了個時間表，到五月一日自己生日當天，若阿 May 還不肯回心轉意，他就會吃掉所有四月三十日到期的菠蘿罐頭。到了五月一日那天，遭拋棄的金城武只好吃掉所有過期罐頭。
王家衛的時間作惡劇還應包括這個：五月(May)，一夜之間被兩個同叫阿 May 的女人拋棄，金城武便決定以後「不再找叫阿 May 的女人」。
在王家衛的電影裡，常有時鐘的鏡頭出現。即使在沒有對鐘的古代，王家衛總有辦法。《東邪西毒》，王家衛便找來張國榮這個長舌黃曆，絮絮不休的敲着更鼓：
「初六日，驚蟄。
「初四，立春。
「十五日，晴，有風
「初十日，立秋，晴」

（都說王家衛・七）

(c)

Figure 8: Three Examples of Multi-lingual Documents

or vertical. Consecutive lines of text are aligned perpendicular to the line stream and proceed downwards in the case of horizontal text and horizontally away from the entry point in the case of vertical text.

Thus, for example, a frame whose entry point is left and whose line stream is horizontal corresponds to horizontal lines of text with each line being read or written from left to right (away from the entry point) and with the lines themselves ordered vertically downwards, of which English text is an example. Arabic, which is written horizontally from right to left, has the same horizontal line stream but its entry point is right. Traditional Chinese text is written vertically in columns with the columns progressing from right to left, so it has a frame whose entry point is right but whose line stream is vertical. And Mongolian, which is written in columns which progress from left to right, also has vertical line stream but its entry point is left.

We further define the *rotation* of a frame to represent the rotation (if any) of the characters of the text it contains. For simplicity we take this to be some multiple of 90°, though again it

would be very simple to extend the model to allow arbitrary rotations.

The contents of a frame can be either ordered or unordered, or may indeed consist of an ordered part and an unordered part. For example, each of the three documents illustrated in Figure 8 is totally ordered because the position of each character is determined by the characters preceding it, whereas a multi-lingual newspaper might be considered as being a totally unordered collection of logically unrelated newspaper articles, the order of these being determined solely by the people producing the newspaper and being alterable at will.

The ordered part of a frame is essentially a string of text (characters) which, as we have seen, may have other pieces of text with different reading and writing directions and with rotated characters embedded in it. These embedded pieces are themselves frames, however, so generally the ordered part of the contents of a frame is a list of *tokens*, each of which is either a character or a frame.

The unordered part of the contents of a frame represents objects which have no logical connection with the ordered part or with each other and which can be positioned at will when the document is displayed or printed. Each of these is just a frame, and in terms of the display or printing of the document represents an area of the frame which its ordered component is not allowed to fill. These "holes" may either contain text themselves or may simply be empty, perhaps representing the location of images. We refer to these frames as *free frames*.

4 A Formal Model of Multi-lingual Documents

Based on the analysis presented in Section 3, we go on to develop a formal model which describes general multi-lingual documents. We use the RAISE specification language (RSL; [1]).

In RSL we can define the properties of a frame (entry point, line stream and rotation) via type definitions:

type

Entry_Point == left | right,

Line_Stream == horizontal | vertical,

Rotation = {| r : **Nat** • r ≤ 3 |}

Here, the first two types are *variant* definitions which simply list all possible values of the type. Rotation is defined via a *subtype*, in this particular case a subtype of the natural numbers (non-negative integers) **Nat** which includes only the numbers up to and including 3, each number

representing a rotation of the characters from the normal way they are written of that multiple of $90°$ in an anticlockwise direction[6].

We can then combine these into another type, Orientation, which specifies all the properties related to the directionality of a frame, using a *record* type:

type Orientation :: entry : Entry_Point stream : Line_Stream rot : Rotation

A frame itself is then composed of an orientation, a string of text and a list[7] of its free frames representing its ordered and unordered contents respectively, and other information, such as the name and size of the font and the typeface and the colour in which its contents are to be displayed. This last information is collected together into the type FormatInfo, the definition of which is deliberately left incomplete to allow other information to be added to it if necessary.

Text is made up of *tokens*, each of which may be either a *character* or a frame, as represented by the *union* type in the definition below. The type Character is defined as a *sort*, which represents an abstract type, the precise details of which are unimportant for the specification, but this will clearly include the character sets of the various languages we wish to include in our documents as well as various formatting (control) characters used for defining the structure and the layout of the document when it is printed or displayed.

type

Frame ::
orient : Orientation format : FormatInfo string : Token* freeFrames : Frame*,

Token = Character | Frame,

FormatInfo = font_name $\times$ font_size $\times$ typeface $\times$ colour $\times$ ...,

Character

Finally, a document itself is represented simply as a frame[8]:

type Document :: frame : Frame

[6]Note that we could extend this to arbitrary rotations by instead defining the type Rotation as a subtype of the real numbers from zero up to 2π.

[7]Although this means that the free frames are in fact ordered in the model, this order is used purely as a means of defining functions such as the indexing functions given later in this section. The order is ignored when the contents of the frame are displayed as is explained in Section 5.

[8]Note that, although we have not included the possibility of hypertext links in this model, we believe that only a simple extension would be required to describe them, probably a mapping from frames to frames.

We can further extend this basic formal model to include formal definitions of functions associated with documents, for instance the "position" of characters and frames within a document. To illustrate this, we introduce two new type definitions, Nat_1 and Index. The first of these represents all the positive integers (i.e. not including zero), and the second is simply arbitrary lists of such numbers (including the empty list).

```
type
    Nat₁ = {| n : Nat • n ≠ 0 |},
    Index = Nat₁*
```

In computer terms, the *index* may be thought of as representing the current cursor position within a document, with the first element of the list representing the position within the string or free frames of the frame representing the whole document, and the rest of the list, if any, representing the index within a subframe if the first element happens to indicate a frame rather than a character.

A given index is said to be *valid* in some document if it corresponds to the position of some character or frame within it. This notion is represented in RSL by the following series of functions (which are called *values* in RSL):

```
value
    is_valid_index : Document × Index → Bool
    is_valid_index(d, i) ≡ is_valid_index(frame(d), i),

    is_valid_index : Frame × Index → Bool
    is_valid_index(f, i) ≡
      if i = ⟨⟩ then
        true
      else
        let ⟨h⟩ ^ t = i, c = string(f) ^ freeFrames(f) in
          h ≤ len (c) ∧ is_valid_index(c(h), t)
        end end,

    is_valid_index : Token × Index → Bool
    is_valid_index(t, i) ≡
      case t of
        Token_from_Frame(f) → is_valid_index(f, i),
        _→ i=⟨⟩
      end
```

We can define the "next" cursor position in a similar way using the function next_index.

```
value
        next_index : Document × Index → Index
        next_index(d, i) ≡
            let i′ = i ⌢ ⟨1⟩ in
                if is_valid_index(d, i′) then i′ else
                unstack(d, i) end end
```

This looks first for substructure at the current location (i.e. looks to see if the current index represents a frame), then for structure which continues the current structure. If both of these fail it means that the current index represents the end point of some structure and the next index is then found by regressing to the containing structure and trying to extend that in the same way, and so on recursively. This is done using the auxiliary function unstack, which is defined in terms of other auxiliary functions increment_last and remove_last as follows:

```
value
        unstack : Document × Index → Index
        unstack(d, i) ≡
          if i = ⟨⟩ then ⟨⟩ else
            let i′ = increment_last(i) in
              if is_valid_index(d, i′)
              then i′
              else unstack(d, remove_last(i))
              end end end,

        increment_last : Index → Index
        increment_last(i) ≡
          case i of
            ⟨h⟩ → ⟨h+1⟩,
            ⟨h⟩ ⌢ t → ⟨h⟩ ⌢ increment_last(t)
            end
          pre i ≠ ⟨⟩,

        remove_last : Index → Index
        remove_last(i) ≡
          case i of
            ⟨h⟩ → ⟨⟩,
            ⟨h⟩ ⌢ t → ⟨h⟩ ⌢ remove_last(t)
            end
          pre i ≠ ⟨⟩
```

The function increment_last simply adds 1 to the last number of some index, while remove_last removes the last number.

Note that this process of determining the next index is cyclic – if we have reached the end of the whole document the next index is defined by these functions to be the beginning of the document.

We now go on to discuss the design of a software system which supports this model.

5 Software Support for Multi-Directional Multi-Lingual Texts

The most fundamental requirement on a "MultiScript" software system supporting fully multi-lingual texts is that it should be possible to use it to create, modify and display multi-lingual documents such as those illustrated in this paper.

One way of achieving this would be to design a completely new system based around the model presented in Section 4, but this is somewhat wasteful because there are already many systems, like the SUDAR system described in Section 2, which offer good support for certain languages both in terms of providing fonts covering the appropriate character sets and in providing a good method of inputting those characters. Indeed, although each tool in itself may only support a small set of different languages, taken as a whole they cover a very wide range. The main drawback of these systems is that they generally do not support the whole range of different reading and writing directions and character rotations which are required to support truly multi-lingual documents. Indeed many insist that all the languages they support should be written and read in the same sense, and they can thus be said rather to support only the different alphabets (or character sets) of the languages rather than the languages themselves.

Another drawback is that many of the tools use different encoding schemes for their character sets, which makes them mutually incompatible as they stand. Neither do the majority of them provide support for document processing (e.g. the definition of the structure of the document) as opposed to simple text processing.

Rather than trying to re-implement useful functionality that is already available in existing tools, however, the MultiScript system should provide an environment within which a wide range of existing tools can be integrated and which in addition allows the reading and writing directions of pieces of text to be changed and permits the rotation of the characters it contains. More specifically, it should allow lines of text to be written vertically, with the lines ordered top to bottom, or horizontally, with the lines ordered either left to right or right to left, and it should permit characters to be displayed either rotated 90° in either direction or upside down[9]. It should also allow both the font associated with some piece of text and the size of the frame in which the text is displayed to be changed, and perform any hyphenation needed as a result of this reformatting[10].

[9]Although none of the examples of multi-lingual documents we have studied actually contains inverted characters there seems no reason not to support them.

[10]This might possibly be done using appropriate parts of the existing tools for the appropriate languages.

The system should also provide its own functions for editing and creating multi-lingual documents and for displaying them. We deal with these separately below.

5.1 Creating and editing documents

The MultiScript system should provide two different ways of entering text into a document as illustrated schematically in Figure 9. First, complete sections of text written in different languages using existing text processing tools can be inserted into a document. These sections might be stored in files, in which case the contents of the file could be read into the MultiScript system directly. Alternatively, the required text might be currently displayed on the screen in another window in which an existing text processing tool is running. In this case, the text could be copied directly onto a clipboard associated with the MultiScript system and then pasted from there into the appropriate location in the document.

In both of these methods the imported text may be coded using a different character encoding scheme to that used by the MultiScript system itself provided the font needed to display the text either is already available in the MultiScript system or can be imported into it, which is possible if it has the same type as the fonts used by the MultiScript system.

The imported text would form the contents of a separate frame within the MultiScript text, the font of this frame being thus the same as that used by the tool which was used to generate the text. This ensures that the text is displayed correctly in the MultiScript system.

The other way of entering text into the MultiScript system would be directly via the keyboard. To enable the characters of different languages to be entered in this way, the MultiScript system should incorporate a tool which allows the user to define the mapping between the keys on the keyboard and the characters in a font. In addition, this tool should allow the user to define "context rules" by which a given generic character is replaced by a variant form in certain situations. In this way, all the different variant forms (for example initial, medial and final) of a character can be bound to a single key or at least to the same sequence of keystrokes on the keyboard and the context rules can be used to determine which specific variant should be used for each occurrence.

5.2 Defining the physical layout of documents

We define the physical layout of the text in a document in terms of *viewports*, where a viewport represents some rectangular area on a computer screen, on a piece of paper, or on any other display medium.

The MultiScript system should allow the user to associate viewports with frames of text in two different ways depending on whether the frame is part of the string of text of its containing frame or is free therein, that is whether it belongs to the ordered or the unordered part of the frame's

contents. In the first case, each frame would be associated with exactly one viewport and the user should be able to define one dimension of that viewport, namely that determining the maximum length of the lines it can contain. This length is the vertical height for frames whose line stream is vertical and the horizontal width for frames whose line stream is horizontal. The other dimension of the viewport would be determined by the system and should be the minimum necessary to accommodate the contents of the frame. The viewport should be completely contained within the viewport of the containing frame (if any), so it should be continued automatically on subsequent lines of the text within the containing viewport where necessary.

A viewport of this form is termed an *unbounded* viewport. Entering text into an unbounded viewport is illustrated in Figure 10, which shows the display of a frame whose contents L consist solely of an ordered part, this comprising a string of characters A, a frame B, another string of characters C, and a frame D. The regions of the display P which each of these parts generates is given alongside the diagram.

When the unbounded dimension of the viewport reaches the boundary defined by the containing viewport the viewport is automatically continued on the next line (in the sense of the containing viewport's text), the continuation having the same fixed dimension as the original viewport. Thus, in the example, the text inside the frame B is too long to be displayed in the area B_1 with the given (vertical) line length so it automatically continues into the area B_2. This is positioned according to the sense of the text of the frame A within which the text B is embedded, and it has the same height as the viewport B_1. This process can continue over several lines if necessary, as illustrated for the text of the frame D which requires three viewports D_1, D_2 and D_3.

The viewport associated with the top-level frame of a document is considered as being unbounded, and so to behave in exactly this way, even though there is no containing text.

The second method of defining a viewport applies to the free frames contained within some frame. In this case, the user may associate a sequence of one or more viewports with each frame, all but the last of which should be *bounded* viewports, that is having their position and both their dimensions defined by the user. The last viewport in the sequence may be either bounded or unbounded, but in the latter case would also have its position defined by the user. Again, all these viewports must be contained within the viewport of the containing frame, so any final unbounded viewport should be automatically continued over as many lines of the text of the containing frame as necessary.

The position of a viewport associated with a free frame can be defined by giving the Cartesian coordinates of the top left-hand corner of the viewport relative to the top left-hand corner of the viewport of the containing frame as illustrated in Figure 11.

The text contained within the given free frame is assumed to fill the sequence of viewports associated with that frame in turn. If all of these viewports are bounded then the space available may not be sufficient and the tail of the text may not be visible.

When text is being entered into a bounded viewport, no continuations should be generated

automatically. When the viewport is filled, the text following should remain invisible until a continuation viewport is designated by the user.

The contents of a frame should be displayed as follows. First, the fixed dimensions of the viewport are delineated. Next, the viewports of the free frames are positioned and their dimensions delineated. If the final viewport in the sequence associated with one of these frames is unbounded then the contents of that frame have to be displayed in order to determine its dimensions. Finally, the string of text associated with the frame is displayed in the space remaining, that is inside its viewport but avoiding the spaces allocated to the viewports of its free frames. This is illustrated in Figure 12, where A represents the string of text of some frame and B and C are free frames within it.

The MultiScript system should not only allow the user to allocate viewports to frames in a document as described above but should also store the positions (where appropriate) and fixed dimensions of these viewports along with the text of the document when an editing session is saved so that the layout of the document can be reproduced. The system should also allow viewports associated with free frames to be repositioned and any viewport to be resized.

The effect of repositioning a viewport associated with some free frame is simply that the ordered part of the contents of the frame which contains that free frame should be reformatted within the containing frame's viewport so as to take account of the change in the area that is available to it. The result of this should of course be identical to that obtained by redisplaying the whole contents, ordered and unordered, of the containing frame.

The effect of resizing a viewport depends on whether it is bounded or unbounded. If the viewport is unbounded then the text in the viewport is simply reformatted to fill the new line length and the unbounded dimension is adjusted accordingly. If it is bounded, any text which can no longer be accommodated within the viewport after reformatting should be moved to the beginning of the next viewport, if any, in the sequence associated with the given frame, the text of which is then reformatted similarly, and so on until the end of the sequence of viewports is reached. If there is no next viewport the extra text will not be visible. If on the other hand reformatting the text in a viewport creates space, text should flow in from the next viewport in the sequence to fill the space, then the next viewport will itself be reformatted in the same way. However, the dimensions of a viewport which contains other viewports can only be reduced as far as is permitted by the constraint that these viewports must remain inside it.

6 Conclusions

We have presented the results of an extensive analysis of multi-lingual documents, showing how these can consist of pieces of text with different reading and writing directions and constructed from character sets which may be rotated. On the basis of this, we have given a formal model of a generic multi-lingual document and shown how standard functions of text processing systems, such as the position of a cursor, can be defined in the model using functions. We have further

described the sort of software support which is already available for many languages using the Mongolian SUDAR system as an example, and we have indicated how these existing systems could be integrated within an environment which additionally supports texts whose directionality changes and which incorporate rotated characters.

The work presented here represents the results of the first phase of UNU/IIST's 'MultiScript' project, which began in the last quarter of 1995. In the next phase we hope first to formalise the requirements outlined in Section 5 in RSL, then to go on and implement a system which fulfils these requirements.

Acknowledgments

Thanks to Chris George for helpful comments on a draft of this paper, and to Dines Bjørner, Chris George and Oliver Corff for many useful discussions during the course of the work on which it is based.

References

[1] The RAISE Language Group. *The RAISE Specification Language.* BCS Practitioner Series. Prentice Hall, 1992.

[2] I. J. Schmidt. *Grammatik der mongolischen Sprache.* 1831.

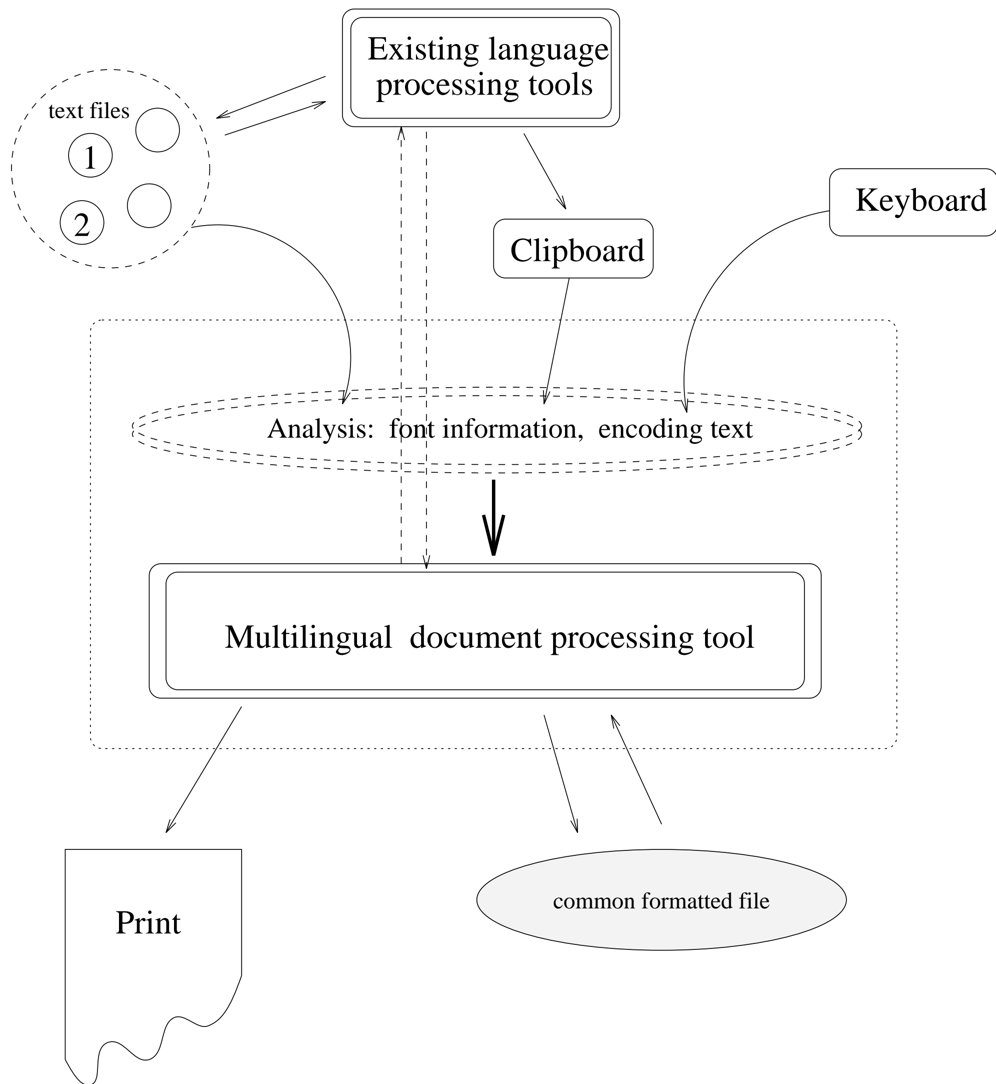


Figure 9: Entering Text into the MultiScript System

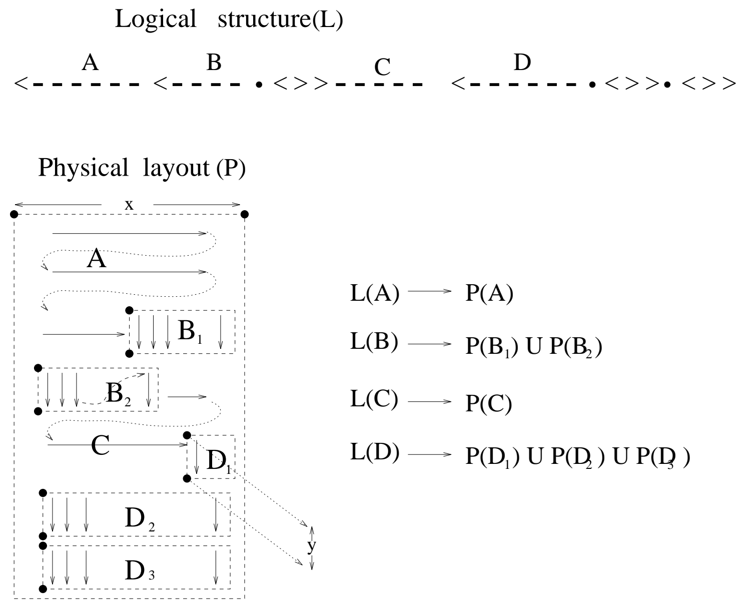


Figure 10: Entering Text into an Unbounded Viewport

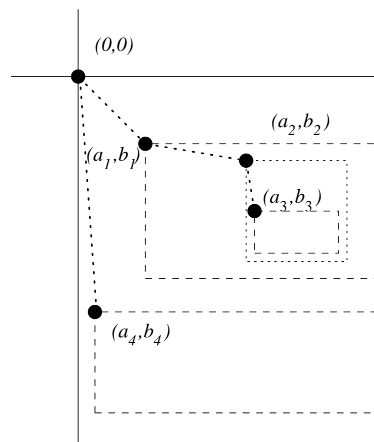


Figure 11: Positioning Free Frames within a Viewport

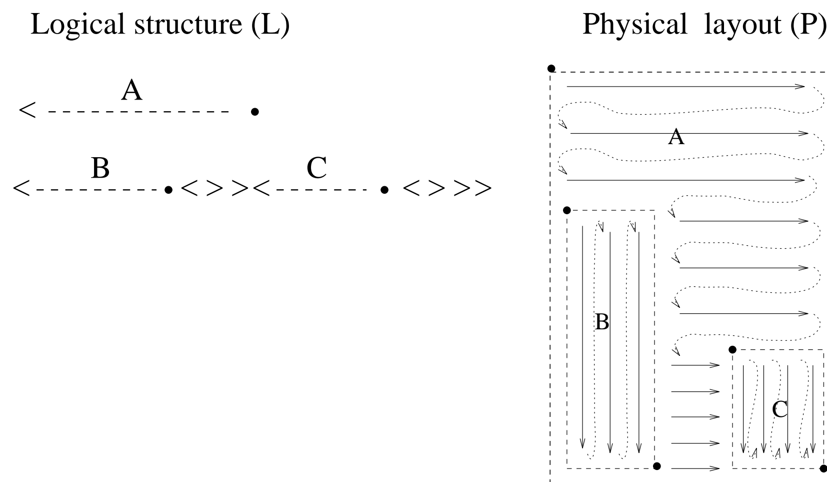


Figure 12: Displaying the Contents of a Frame