システムソフトウェアと
オペレーティング・システム 68－4
（１９９５．３．２４）

Codeset Independent Full Multilingual Operating System: Principle, Model and Optimal Architecture

Yutaka Kataoka*, Tadao Tanaka†, Tomoko Kataoka*,
Kazutomo Uezono†, Shoichiro Yamanishi† and Hiroyoshi Ohara†

* Centre for Informatics, Waseda University
† Ohara Laboratory, School of Science and Engineering,
Waseda University

Abstract

Localization based on POSIX Locale Model does not satisfy the Internationalization (I18N) and has inhibited computability, interoperability and information exchangeability. An I18N environment not only satisfies ISO 2022 and ISO 6429 but also permits mixing all character code sets including thier extensions for I/O and Text Manipulation/Communication without dependending on any orthography of a language. Thus, it is clear that I18N must be realized as a Multilingual system. To realize the system, all writing scripts and their orthographies were analyzed, and essential informations were discovered for defining Character Set and Final Glyph Set, and for specifying Final Glyphs. And beyond ISO specifications, extensions of character codepoints were generalized. By using those informations, Global IOTMC Model that ensures computability, interoperability and information exchangeability was established. Then, Meta Converter System that was designed and coded optimally realized generalized/unified mechanism of the Multilingual System.

片岡 裕*、田中 忠雄‡、片岡 朋子*、上園 一知‡、山西 正一郎†、小原 啓義†

* 早稲田大学 情報科学研究教育センター
‡ 早稲田大学大学院 理工学研究科
† 早稲田大学 理工学部

概要

POSIX Locale に基づくLocalizationは、国際化に対応せず、情報交換可能性と計算可能性を著しく阻害する。国際化環境とは、単にISO 2022やISO 6429を満足するのみならず、固有の拡張を含めた全情報交換用符号集合を混在させ、全言語の正書法に依存することなく、入出力・テキスト処理・プロセス間通信を可能とするMultilingual Systemであることが必須である。これを実現するため、全世界の文字と正書法を調査し、文字集合定義・最終表示図形集合定義・表示図形決定の各必須情報を求め、さらにISO規格とそれを越える情報交換用符号集合固有の拡張を一般化した。この情報を用いてGlobal IOTMC Modelを構築し、計算可能性・情報交換可能性を保証し、最適化したMeta Converter Systemを作成し、単一の構造で一般化された機構によるMultilingual Systemを実現した。

1. Introduction

Truly internationalized computing requires, at least, allowing simultaneous mixing any number of scripts for any language, and use of any number of code sets (and extensions thereof) for any script. By contrast, most approaches to *Internationalization* (I18N) use the *POSIX Locale model* [1,2] which localizes each application to be at most bilingual, and leaves the code-set handling poorly defined. Since POSIX, ISO C[3] and Amendment 1: C Integrity (Draft)[4] allow defining Multi-code-set (MCS) locale, it is possible to apply a MCS locale. But the MCS locale reveals contradictions of the Locale model[5,6]. Also POSIX and C do not allow for *Non-Fixed-Length* (NFL) codepoints such as those given by ISO 6429[7] or by IS 13194[8]. Thus, POSIX and C do not meet the minimum requirements for true I18N[9]. Nor do they provide for orthography independence, language-independent text manipulation and communication.

X11R5 implements partial I18N, with limited MCSs[10]. But in X11R6, the OM and IM are hard-coded to the locale. Neither R5 nor R6 ensure computability and the locale model prevents them from providing true I18N. Nor do they provide for data exchangeability.

True I18N systems must provide Multilingual Input, Multilingual Output, Text Manipulation, and Text Communication; with all components ensuring computability, interoperability, and information exchangeability, with unlimited MCSs. All functions should work consistently with character code extensions[11] (in IS 13194 for a set of *Indian scripts*, 94 codepoints is extended to more than 2000 real characters). Thus, an I18N system must be *Multilingual* – code set and language independent with the codepoint extensions.

To implement our system, all writing scripts and their orthographies were analyzed, and essential information was discovered for defining *Character Set* and *Final Glyph Set*, and for specifying *Final Glyphs*. Using this information, we generalized character codes beyond ISO specifications, to a *Global IOTMC Model*[6] that ensures computability, interoperability and information exchangeability. The Model provides the *Meta Converter System*[6,12] that was optimally designed and coded, and the System realized a generalized and unified mechanism of the Multilingual System.

2. Scope of Multilingual I/O and TM/C System

A Multilingual I/O and TM/C system should have four components: Input, Output, Text Manipulation, and Interprocess Communication. Each component must allow for 1) simultaneous handling of unlimited sets of MCSs, 2) ISO 2022[13] specifiability (even for control character sets), 3) ISO 6429 codepoint extensions, NFL codepoints (e.g., TIS 620-2533[14] – *Thai script*, IS 13194 a set of Scripts in India), 5) conversions between WC and mb, which contains *Control Character Sets*, *Control Sequences*, NFL codepoints, and information about *direction dependency* and *position dependency*, 6) interactive selectability from a set of MCSs locales, 7) *Text Manipulation Codes* (TMCs) for independence of text manipulation functions from code set and language/orthography, and 8) information for interprocess communication for each application. Any Multilingual I/O and TM/C system must satisfy the above requirements (i.e., code set independence by WC converted from mb by obvious rules and language/orthography independence by use of TMCs). As a result, each codepoint in each mb/WC/TMCs must be uniquely convertable with codepoint extension methods among mb, WC and TMCs. Thus, a system that cannot define extension rules beyond ISO to determine each element in each set is not a Multilingual I/O and TM/C system.

3. Definition of Generalized Multilingual I/O and TM/C System

To avoid hard-coding, the system should have data files that describe definitions of all information for the four components described above. And to keep consistency, only one executable process of each component of the four should provide informations and perform functions for all processes. A system satisfying above requirements qualifies as a generalized and unified system.

4. Essential Information for Definition of Character based on Analyis of World's Writing Scripts

To design the generalized multilingual I/O and TM/C system, world's writing scripts and their writing conventions were analyzed and classified[10]. Writing scripts fall into the following categories, 1) *Phonemic*, 2-1) *Conjunct Syllabic*, 2-2) *Pure Syllabic* and 3)

Ideogrammic. This categorization is still not enough to define informations held in a *Character*.

When a character is written, it is called a writing script. The *forms* of characters that writers consider equivalent can vary according to their writing direction and their position in a word. They can also vary in up to four ways depending on their position in a word; 1) *Initial* (first), 2) *Medial* (between first and last), 3) *Final* (last) and 4) *Independent form* (first *and* last, i.e., a single character word). For our purposes, a *character* is rigorously defined as a named equivalence class of triples, each of which consists of a direction, a position, and a *Final Glyph*. For maximum generality, we define all characters as being both 1) *Direction dependent* and 2) *Position dependent*[6,11]. Thus, each of our characters represents a set of 16 final glyphs (4 directions × 4 positions) (Fig. 1. Notice 'Hello!' and punctuation marks). In some characters, some of the 16 final glyphs happen to have the same shape. Therefore, a Character should have informations to select one final glyph – *Final Glyph Information* that is the informations held in a Character.

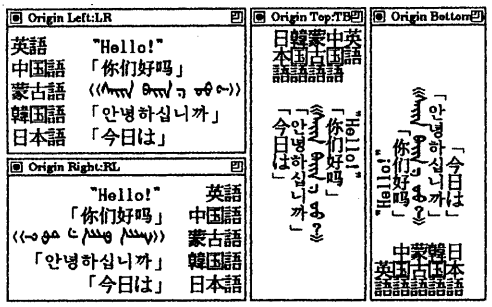

Figure 1

The Final Glyph infomation must contain 1) *Direction Dependency* and 2) *Position Dependency*. The Direction Dependency information consists of a drawing direction of a character (a character has a fixed direction or it should be written according to a current writing direction) and of selection rules of a glyph when a glyph shape varies according to each direction. Position dependency information specifies selection rules of one glyph out of 4.

As a default, *Drawing Time Information* that contains *Origin of Line* to specify a direction of word progress must be defined to select one correct Final Glyph.

5. Definition of Informations for Character Code Set based on Analyis of Code Set Designs

Character code sets are classified into Graphic Character Set (GCS) and Control Character Set (CCS). It is clear that a GCS lists 'Characters' that hold Final Glyph Information, e.g., ISO 8859-6 for Arabic lists only Independent Forms. If characters in an Arabic word were written in all Independent Forms, it is impossible to read – such a sequence means 'Number' in Arabic because Arabic script is a *logograph*. So it is natural to think that ISO 8859-6 lists Characters[15]. Except for a few cases, all codepoints in all GCSs are defined as Characters, i.e., each codepoint is defined as a name of a set of Final Glyphs – *Name Definition*. When a codepoint in a GCS is directly defined as a final glyph, the codepoint is called as *Glyph Definition*.

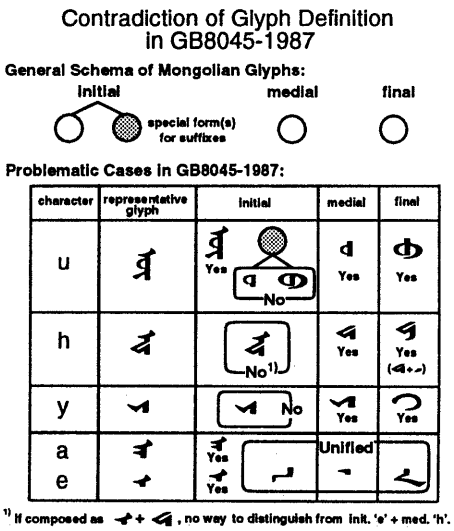

Figure 2

If a GCS contains both a codepoint as Name Definition and its respective codepoints by Glyph Definition, such a GCS provides contradictions. ISO 10646 contains 5 codepoints *Theh* in Arabic, 1 as Name Definition and 4 as Glyph Definition, with no description for the relations between Name Definition and Glyph Definition. Thus, ISO 10646 cannot be used for text processing by code set independent applications. GB 8045-1987 for *Mongolian Script* was designed as a mixture of Glyph and Name Definitions. The code set does not explain any extension ways to define relation among Name Definition and Glyph Definition codepoints (note

that one page is even lacking). By an analysis of Mongolian Script orthographies, it is impossible to define Final Glyph Information for GB 8045 (Fig. 2) – there is no way to add information to each Character by unification of Characters by their shapes.

Adding to code extension techniques of ISO 2022, a codepoint can be extended by combinations of codepoints in GCSs or by definitions of ISO 6429. To treat generally both GCSs and CCSs for codepoint extensions, it is possible to define that a codepoint has two roles: one is that a codepoint can specify a Final Glyph Set, and the other is that a codepoint invokes *Control Functions*. Defined as this, a codepoint in a CCS does not have a rule to specify a Final Glyph Set, and the same codepoints in GCSs have both roles. Thus, information in a codepoint can be defined as specifying these two roles.

To generalize *Codepoint Extension Methods*, analysis of international/national code set designs is also essential. Code set designs are classified into the following categories, 1) 1 codepoint for 1 Character, 2) 1 codepoint for multiple Characters, 3) multiple codepoints for 1 Character and 4) multiple codepoints for multiple Characters. By the analysis above, the number of codepoints in a GCS or in a CCS is less than the number of those in an extended set of GCS or CCS. Therefore, *Codepoint Extension Rules* must be added to the GCS or the CCS definition to use a GCS or a CCS correctly. After the extension, a GCS or a CCS specifies a set of real codepoints that have two roles above. A Character code extended by the rules from a GCS (CCS) is called an Extended GCS (Extended CCS). Thus, codepoints in a WC must be converted from all codepoints in all Extended GCSs and in all Extended CCSs.

IS 13194 requires multiple different rule sets of extensions corresponding to a set of writing scripts derived from *Brahmi Script*. To process simultaneously all sets of scripts derived from Brahmi Script, interactive Codepoint Extension Rule set selection that is out of ISO 6429 must be realized. Thus, the mechanism to specify a combination between GCS and its rule set(s) is essential.

A code set that does not permit to define computable Codepoint Extension Rules is called *Miss-design code set* – ISO 10646 is a typical one.

6. Definition of the Relations among mb, Extended Set, WC and Final Glyph Sets

Mb is defined as a set of GCSs and CCSs. Then each GCS (CCS) is extended to its Extended Set with Codepoint Extension Rules belonging to the GCS (CCS). Note that it is possible to combine a codepoint in a GCS (CCS) and that in different GCS (CCS).

Since a codepoint of an Extended Set stands for Character, it can be mapped to a Final Glyph in a Final Glyph Set with Final Glyph Informations. Since a codepoint corresponding to a Character has two roles, a codepoint from a CCS can be mapped to a Control Function. Thus, a set consists of a set of Final Glyph Sets and that of Control Functions contains all Final Glyphs and all Control Functions as the elements of the set. The set must be defined as WC to keep computability of a system – the set is the biggest and involves all informations of GCSs and of CCSs. WC must be one in a system to keep consistency among GCSs and CCSs. Since reverse conversion from codepoints of WC to that of mb should be done, informations for the conversion must be held in the codepoints in WC.

Therefore, a codepoint of WC should have informations 1) to specify a Final Glyph (if the codepoint has a glyph), 2) to specify a Control Function (if the codepoint has a Control Function), 3) to enable reverse conversion to mb and 4) to correctly locate the codepoint in a sequence of WC codepoints at drawing time (*Origin of Line Progress* and Origin of Line). From the information to specify a Final Glyph, informations to specify a index of a font in a font file and ID of the font file can be obtained, so any design and any number of font files can be used for WC.

Having those informations in a codepoint of WC, the codepoint is uniquely defined through all locales. Thus, WC defined here can be used as locale independent and it is independent from code sets in mb. The width of the WC is determed by the informations - It is clear that 16 bit WC cannot retain them.

Since POSIX does not suppose any extension of codepoint in mb, some functions do not always work, e.g., wctomb does not work for the permutation of two codepoints in IS 13194.

7. Introducing TMC and Generalized Text Manipulation Functions

As defined above, WC does not have enough information for text manipulation. Nor WC ensure to stand for one character to be processed. There are multiple ways to normalize to one character, although WC cannot be used for text manipulation. Note that not always a proccess unit matches a Character.

To generalize text manipulation in a multilingual text, a code that is normalized by a unit to be processed by absorbing language and orthography differences – *Text Manipulation Code* (TMC) is required. To avoid reducing computability, a mechanism that provides multiple TMCs from WC by different normalizations must be provided. By our researches, that basic unit to be processed was discovered, and so were the basic functions to manipulate text strings and techniques to generalize the functions to TMCs.

One codepoint of TMC has a *Character-ID field* and an *Attribute field.* The Attribute field retains bits of categories of a character, such as punctuation symbol, phonemic or position dependency. Those informations should be definable and described in WC-TMC conversion tables. Especially, TMC-ID 0 is provided by our system which is normalized by a character for the purpose of basic multilingual text manipulation.

By the informations in the Attribute field, text manipulation functions could be designed as TMC independent, i.e., the text manipulation functions use both or one field for the purposes.

8. Supplying Information for Multilingual Interprocess Communication

In interprocess communication, ISO specifications do not supply informations for a locale. And locale dependent factors are all implementation dependent. Thus, a locale dependent system is closed. Therefore, it is clear that a model independent from Locale model is required.

For interprocess communication, a multilingual system should return the following informations; Designation Sequences from names of GCSs/CCSs, a GCS/CCS name from a codepoint of WC, Invocation sequences corresponding to invocation functions in given CCSs, control character sequences corresponding to Control Functions in given CCSs and any current status of *In-use Table* and *Intermediate Table.*

It is essential not only to return the informations but also to associate a code set with a rule set to generate WC and other code sets – IS 13194 can be used for all different scripts derived from Brahmi.

9. The Multi-Locale Model and the Global IOTMC Model

Two models were established that satisfy conditions and definitions described above. The Multi-Locale Model provides a set of MCS locales by *OpenLocale* function that returns each Locale-ID. By using of the ID, MCS can be gotten I/O simultaneously with functions provided anew. Also the function *setlocale* can interactively select a MSC locale in the set of locales, and the setlocale function permits I/O and locale-related functions just as described in POSIX and ISO C. Since POSIX and ISO C are subsets of this model, it ensures backward compatibility. Only one WC through all locales is provided. Thus, it is possible to share WC codepoints among different locales. But TMC-ID 0 is provided only corresponding to a set of locales specified by OpenLocale function. Also informations for interprocess communication are restricted within a range of code sets specified by the locales.

On the other hand, the Global IOTMC Model covers all of I/O and TM/C. This model provides – without limitation of locales – TMC-ID 0, basic text manipulation functions and interprocess communication functions. It is not necessary to open locales in this model. Once calling *InitGlobal* function to initialize, all graphic character sets and control character sets can be used without a locale-ID. And this model also provides a switching mechanism that associates a graphic character set and its Codepoint Extension Rules (this mechanism solves IS 13194 and Perso-Arabic Scripts problems). By the optimal implementation of the architecture of the models, all functions of the models can work simultaneously.

10. The Architecture of the Models

The architecture of the models is based on the *Meta Converter System* that converts encoding schemes, Trans-character-set converting GCSs and CCSs to WC, and converting them to TMCs by *Trans-Unit conversion* via WC, since all sets in the system were defined clearly

to keep computability. The system generates WC and IWC (*Internal Wide Character* indexing only fonts in font files) from mb by the rules described in the tables. And the system generates TMCs from WC. The system also converts TMCs to WC when conditions for reverse conversion are satisfied.

The Meta Converter System is a complex of automata and each automaton is generated by the *Meta Converter Table Compiler* that compiles data definition tables. By analyzing writing scripts and code sets, optimal paths and structures of the automaton complex and each automaton were discovered. To minimize the amount of code by the compiler and to make it faster, basic extension functions were discovered and implemented. Thus, the automata call the functions with rules that are also generated by the compiler. Therefore, the best performance is ensured. All common areas generated by the compiler is shared by the processes.

Each functional part is designed as a module that calls the Meta Converter System and the System returns all informations for all functions of the I/O and Text Mipulation/Communication modules. Thus, each element of a set is ensured as unique. By this architecture, the total system can keep computability, interoperability and data exchangeability.

11. The Implementation of the Meta-Converter System

The Meta Converter System consists of *Meta Converter Table Compiler*, *Trans-Unit Converter*, *GCS/CCS Information Functions*, *Character Information Functions*, *Text Manipulation Functions*, and *Interprocess Communication Assisting Functions*. The encoding scheme converter and the trans-character-set converter are parts of Trans-Unit Converter. All converters above are collectively called as Meta Converter.

11.1. The Meta Converter Table Compiler

The Meta Converter Table Compiler (MCTC) compiles data tables and generates automaton bodies that are loaded and used by the Meta Converter System. There are three categories of data tables, 1) Relation Tables describing *GCS Tables* and *CCS Tables* to be loaded by the compiler, 2) GCS Tables and CCS Tables specifying elements of WC, and 3) *TMC Tables* describing conversion from/to WC.

In Relation Tables, locale, multi-locale and global informations are specified. For one GCS, by different extension rules, multiple GCS Tables can be defined.

The compiler reads the Relation Tables, the GCS/CCS Tables and the TMC Tables, then the compiler generates encoding scheme automata, Trans-code-set automata, trans-unit automa and TMC automata. The structure of each automaton is pointers to predefined optimal functions with calling parameters for minimum storage size and for portability of source code. Each automaton can be selected by merely changing pointers. Thus, after compilation, all automata can be invoked with no locale or GCS table dependencies. These automata are loaded when the initializer in our library is executed.

11.2. The Meta Converter

The Meta Converter consists of the mb/WC Converter, the WC/TMC Converter, the mb/IWC Converter and the WC/IWC Converter. The mb/IWC and WC/IWC Converters are used in the OM.

The mb/WC Converter processes mb through the following steps, 1) Control Functions and Determinative processing, 2) Direction determination and 3) Position determination[11]. In the third step, informations about wctypes functions are stored in a bit field in WC for compatibility with POSIX – note that those functions do not always work correctly. Reverse-conversion executes steps 3 through 1. Since the Control Functions and Determinative processing use a set of functions with rules associating GCS/CCS, and WC retains an ID for the function, it is possible to reverse-convert by a pair of functions and its reverse-function with the same function ID. Thus, with the function pairs, fixed length data types can be used for WC.

Conversion from WC to TMC is done by the WC/TMC Converter through the following steps, 1) generating a TMC Character-ID field from the GCS/CCS-ID, Font ID and Glyph Index of the WC codepoints and rules, 2) generating TMC Attribute field from rules in the converter and 3) generating the rest of the Character-ID field that retains informations to convert TMC to WC. Reverse conversion to WC can be done by using the informations to convert TMC to WC.

Since all conversion paths were determined, the fastest algorithm optimized was implemented with

maximum straight-line code that does not break pipeline stage of CPUs. The algorithm, the implementation and the architecture are described in detail in another paper (in preparation).

11.3. The Character Information Functions

Character Information Functions query and change informations in the Attribute field informations in one element of TMC. Each attribute in the field is stored as one bit. In order to generalize the functions, each attribute name in the field is set in a TMC Table and the functions retain the name and its bit location in the field by data generated by the MCTC. Thus, the functions are called with the attribute name(s) and TMC-ID. The essential functions[9] are as follows, 1) QueryTMCAttribute, 2) QueryTMCInformation, 3) ChangeTMCInformation, 4) QueryTMCCtrlCName. Finding a line separation can be located by checking attributes field defined for the purpose.

11.4. The Text Manipulation Functions

The TMC Text Manipulation functions are also called with TMC-ID. Note that each TMC is independent of all other TMCs.

The basic TMC string functions[9] are 1) TMCStringLength and 2) TMCStringCopy. A TMC string concatenation function can be derived from these.

The essential functions for TMC text manipulation[9] are as follows, 1) InsertTMCString, 2) DeleteTMCString, 3) CompareTMCString, 4) ExtCompareTMCString, 5) SearchTMCString, 6) ExtSearchTMCString, 7) ReplaceTMCString and 8) ExtReplaceTMCString. These functions adjust the Direction and Position Dependency attribute fields.

11.5. The Interprocess Communication Assisting Functions

The GCS/CCS Information Functions are used to inform associations between GCS/CCS and designation sequences, and its extension rule sets. To re-associate designation sequences and extension rule sets, function ChangeGCSExt is used. This function is essential for IS 13194 because ISO does not supply such re-association. Since WC codepoints in a rule set are unique when the rule is changed, it is possible to mix all scripts derived from *Brahmi* simultaneously.

12. Multilingual Application Softwares Realized by the Multilingual I/O and TM/C System

Essential applications were modified to Multilingual by using of the Multilingual I/O and TM/C System.

Adding Multilingual OM calling the Meta-Converter System to X Window System, X Window System could correctly draw any Character in any Code Set in any direction as Multilingual I/O System (Fig. 3). The new X Window System maps a Multilingual Window more than 10 times faster than older systems and new one draws even to several times faster.

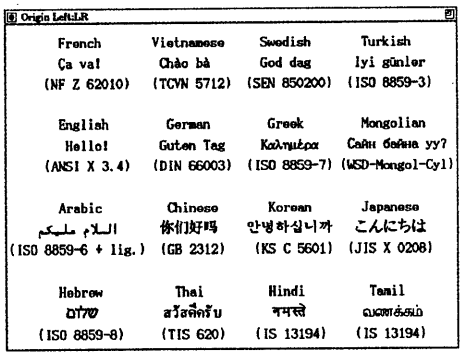

Figure 3

Also the Meta-Converter System provided Interprocess Communication facilities to the X Window System. By using the facilities, the communication between X library and X IM could be defined, and a new X IM is now under development.

Adding to the new X Window System, Multilingual Athena Widget and X Intrinsics were developed. All widgets in the Athena Widget relating to I/O and text manipulation were modified to use the Meta-Converter System. In particular, Multilingual Text Widget could correctly format and edit multilingual texts in any drawing direction and any direction of line progress. The new Widgets permit interactive switching of vertical or horizontal processing.

A Multilingual FORTH system was developed so that it can satisfy Multilingual Common LISP after programming the FORTH. The FORTH calls the Meta-Converter System to process mb/WC/TMCs. The FORTH system compiles source code to run faster. The system has two implementations, a FORTH interpreter with the compiler, and a Runtime library format of FORTH that can be linked from another application.

The library implementation makes it easy to provide complicated application softwares.

Multilingual printing facilities for printers are now on the way by parallel processing ability on networking to the OS.

13. Summary

In our research, requirements, informations and definitions for a Multilingual System became clear. A Multilingual I/O and TM/C system was developed based on the Multi-Locale Model and the Global IOTMC Model. The system was optimally developed based on the Meta Converter System with compiler technologies. Problems with POSIX and ISO specifications were discovered and solved, and problems with some Graphic Character Sets could be defined.

The System can provide code set and language/orthography–independent application softwares – e.g., multilingual editors, text formatters, distributed databases, even Multilingual parsers and syntax analyzers for natural language processing can be provided. Research for those has already begun in our laboratory. By the analysis of characters and orthography, informations to convert a spoken language to Character sequence became clearer. Thus, a new model of a Multilingual parser is now under development.

14. Acknowledgements

Thanks go to H. Daikokuya, K. Maruyama and T. Oya. To Michael E. Turner goes our deep gratitude for his efforts on discussions. Remaining errors, if any, are of course ours.

References

[1] ANSI/IEEE Std 1003.1-1998, IEEE Standard Portable Operating System Interface for Computer Environments (Approved Nov-10,'89 by ANSI).

[2] ISO/IEC 9945-1: 1990, Information technology – Portable Operating System Interface (POSIX) Part 1: System Application Program Interface (API) [C Language].

[3] ISO/IEC 9899: 1990, Programming language C.

[4] ISO/IEC 9899: 1990/DAM 3, *Draft* Amendment 1:1994 (E), Programming languages – C AMENDMENT 1: C Integrity.

[5] Kataoka, Y. et al., A model for Input and Output of Multilingual text in a windowing environment, ACM UIST'91 November 11-13, pp 175-183.

[6] Kataoka, Y., et al., Multilingual I/O and Text Manipulation System(1): The Total Design of the Generalized System based on the World's Writing Scripts and Code Sets, Proceedings of the 49th General Meeting of IPSJ, Vol. 3, September 1994, pp 299-300.

[7] ISO/IEC 6429: 1988, Information processing – Control functions for 7-bit and 8-bit coded character sets.

[8] IS 13194:1991, Indian Script Code for Information Interchange – ISCII, Bureau of Indian Standards, India.

[9] Kataoka, T. et al., Multilingual I/O and Text Manipulation System (3): Extracting the Essential Informations from World's Writing Scripts for Designing TMC and for the Generalizing Text Manipulation, Proceedings of the 49th General Meeting of IPSJ, Vol. 3, September 1994, pp 303-304.

[10] Kataoka, Y. et al., A model for Input and Output of Multilingual text in a windowing environment, ACM Transactions on Information Systems, Vol. 10, No. 4, October 1992, pp 438-451.

[11] Uezono, K. et al., Multilingual I/O and Text Manipulation System (2): The Structure of the Output Method Drawing the World's Writing Scripts beyond ISO 2022, Proceedings of the 49th General Meeting of IPSJ, Vol. 3, September 1994, pp 301-302.

[12] Tanaka, T., et al., Multilingual I/O and Text Manipulation System(4): The Optimal Data Format Converter to/from MB/WC/TMC, Proceedings of the 49th General Meeting of IPSJ, Vol. 3, September 1994, pp 305-306.

[13] ISO/IEC 2022: 1986, Information processing – 7-bit and 8-bit coded character sets – Code extension techniques.

[14] TIS 620-2533 (1990), Thai Character Codes for Computers, Thai Industrial Standards Institiute, Ministry of Industry, Thailand.

[15] Tanaka, T., et al., Generalized Output System that draws multiple code sets on a windowing environment (in Japanese), Proceedings of the 46th General Meeting of IPSJ, vol. 5, March 1993, pp 87-89.