

# 文字定義のこと

守岡 知彦 CHISE Project

## 1 はじめに 文字コードの夢と現実あるいは憂鬱

ここで述べるのは著者が CHISE Project をはじめたに至った経緯と、その後の大部分の時間を費して来た文字データベースいじりの物語!?である。著者が外字問題を認識するに至ったのは確か旧唐書の一節を入力しようと思った時ではなかったかと思う。もっとも当時の私は工学部の電気・電子系の学生であり、べつに中国学を勉強していた訳でもなく、単なる趣味だったはずで、その時は問題を認識するに留まったようだ。あるいは、同じく趣味でかじってたスペイン語を入力するのに必要なアクセント付き文字や記号が足りないということで、外字を作ろうと思った時に ISO/IEC 8859 シリーズを知ったような気がする。また、TRON code って良いのかなと思って調べてみて、実際のコード表が少ししか探し出せなくて悲しかったりした。<sup>1</sup>こんな風にこの頃は良い文字コードやそれが使える良いシステムがどこかにあるんだろうなと思っていた。

### 1.1 Mule の夢と現実

とある山奥にある情報系の大学院に進学してから、現実逃避の一環でかつての文字趣味を思い出したようだ。<sup>2</sup>噂に聞いていた Mule [7] という多言語 GNU Emacs を実際に使ってみたのもこの頃だ。<sup>3</sup>Mule は当時の私にとって夢のようなシステムだった。「国際化」とか「多言語」<sup>4</sup>ができるという「夢の」システム

<sup>1</sup>今思うと、探し出せなかったのではなくて、探し出せた程度しか定義されてなかったようなのだが。

<sup>2</sup>その一環で、MIME 関係のプログラムを改造したり、自作したりした。

<sup>3</sup>もっとも、当時私は Epoch というマルチ・ウィンドウ拡張された GNU Emacs の日本語版を使っていたので、主要な環境にしたのは Epoch 相当の機能が取り込まれた GNU Emacs 19 ベースの Mule が登場してからだった。

<sup>4</sup>「多言語」というべきだが、当時は言語と国の区別がついてない人が少なくなかった。まあ、今でも「多言語」って書く人はいるけど。

の『伝説』<sup>5</sup>はいろいろ語られていたが、実際にいろんな文字が使えて現実に自分が使えるシステムは Mule がはじめてだったのだ。Mule は発展途上のシステムだったが拡張可能性を備えていて実際にいろんな人達がさまざまな文字や言語のサポートを始めた。

このように Mule の利点は拡張可能性にあったといえる。Mule は ISO/IEC 2022 [4] 風の図形文字集合 (“charset” と呼ばれる。以下では、“Mule-charset” と呼ぶことにする) を単位として内部コードを拡張できるようになっていた。また、さまざまな文字コードを定義し、内部コードとの間で変換する仕組みを備えていた。すなわち、使える文字の範囲と情報交換用文字コードの双方を拡張することができたのである。また、簡単な入力メソッドもまた拡張できた。これにより、対象言語のためのフォントと、コード体系やキーボードレイアウト・入力メソッドなどの知識があれば比較的簡単に対象言語をサポートできるようになった。

このように Mule は拡張可能性を備えていたが、漢字においては必ずしも十分ではなかった。Mule の方法では文字を Mule-charset の ID と Mule-charset 内での符号位置の対で表現していたが、このことは本来同じである文字でも所属する Mule-charset が異なれば違う文字と看做されるということを意味する。<sup>6</sup>ISO/IEC 10646-2:2001 [6] の制定より前の時代、最大の漢字数を誇っていた標準として CNS 11643 [14] があり、それが Mule で利用できるようになったことから、<sup>7</sup>一部の漢字マニア達は JIS X0208/0212 に加えて CNS 11643 なども駆使して多漢字文書を表現していたのだが(無論、簡体字がいる場合は GB2312 も)重複符号

<sup>5</sup>オープンソースというものの恩恵の1つは、『伝説』を許さないことも知れない。

<sup>6</sup>当時、Unicode 批判の文脈で、同様の性質を持つ ISO/IEC 2022 の符号切替えの仕組みに関して、こうしたことが利点だと主張する人達もいたが、多分、国や言語や用字系 (script) や書記系 (writing system) といった概念の区別があまり発達・浸透してなかったせいなのだと思う。

<sup>7</sup>当時、花園大学国際禅学研究所にいた Christian Wittern 氏と Urs App 氏は SGML の実体参照形式を利用した多漢字表現の一環として、CNS 11643 の利用を行っていた。

化は結構厄介の種だったように思う。<sup>8</sup>また、このように現実に5万字規模の漢字集合が使えるようになってもやはりない漢字はないというのも現実であって、1文字単位に拡張できない Mule の仕組の不便さが痛感されるようになった。また、いろいろな Mule-charset が追加されていくうちに気がつくとも内部コードの空間が残り少なくなっていて、肝心の拡張可能性も夢の藻屑と消えていたりした。

## 1.2 漢字符号化の夢と現実

### 1.2.1 異体字指向の場合

一方、漢字の性質にあった符号化文字集合の試みというのもあった。例えば、中国の GB では主に簡体字が入っている GB2312 と主に繁体字が入っている GB12345 は対応する文字が同じ符号位置に来るように工夫されている。このような工夫をもっと大規模に行ったものとして CCCII がある。これは  $94 \times 94 \times 6$  文字が 15 個の層をなす構造をしたコード体系で、対応する異体字は各層の同じ位置に置かれるように工夫されている。しかし、異体字が高々 15 個で足りるのかという問題や、異体字の対応関係は実際の所そんなに綺麗じゃないという問題や、異体字を誤って重複符号化してしまう問題などがあり、必ずしも理想の解ではないといえる。符号のビットパターンで異体字関係を表現するというのは工学的に綺麗な感じのする発想で、<sup>9</sup> 計算資源が貧困だった頃はそれが例え近似であってもうれしかったのだろうが、<sup>10</sup> 今となってはそんなに意味がないといえる。要は異体字関係のデータベースがあれば良いのだ。<sup>11</sup>

<sup>8</sup>そもそもテキスト表現にしても表示(フォント)にしても「国」とか「言語」とかと関わりなくごちゃ混ぜにしていた訳で、漢字結合の度合を別にすれば Unicode と気持は似てたのかも知れない。

<sup>9</sup>工学的というより、文字や文字列をついついビットパターンやバイト列として見ちゃう人の発想というべきか。ちなみに、筑波大学の久野氏はかつてそういう種族の人を「バイト君」と呼んではどうかという提案を行っていた。

<sup>10</sup>例えば、JIS X 0208 の第一水準漢字が 50 音順に並んでいるという近似でもそれなりにうれしかった時代があったらしい。

<sup>11</sup>それでも未だに(文脈が若干違うとはいえ)異体字タグとか枝番方式といったものが出て来るというのは、異体字関係をコードとして表現したいという気持の根の深さを意味しているのだろうか？

### 1.2.2 部品指向の場合

異体字指向のコード化と並んで人気のある!?試みとして、部品指向のコード化というのがある。ご存知のように漢字は意符や音符のような部品が偏や傍などとして組合わさって構成されている訳だが、このような構造をコード化すれば漢字の造字力を活かした符号体系ができるのではないかという発想がある。また、ついでに字形合成もできたりすると少ないグリフで多数の漢字に対応できるのではないかということも容易に思い付く。1990 年台の中頃のある時、fj.kanji か fj.sci.lang といったニュースグループ<sup>12</sup>で私を含めた何人かがこの手の話で盛り上がり、私が構造化漢字処理のためのメーリングリストを立ち上げたのだが、割とすぐに盛り下がって、単なる漢字メーリングリストになったような記憶がある。

現在まで続いている試みとしては、1990 年頃に台湾中央研究院の謝清俊氏らが始めた CDP (Chinese Document Processing) データベース [3] がある。これは Big5 と外字 [2] を利用して符号化されており、外字を利用して 14 種類のオペレーターを定義している。そのうち 3 種類は部品の結合を表現したもので、(1) 縦に並べる、(2) 横に並べる、(3) その他 を表現する。また 8 種類の反復記号がある。これは同じ部品を複数個配置することを表現するものである。また、この他に特殊記号が用意されている。なお、CDP データベースの漢字構造表記では入れ子を認めていない。

また、台湾の中華電子佛典協會 (CBETA) では外字表記のために部品組合せの発想の形式を用いている。この方式は Big5 の漢字と ASCII 文字で記述したオペレーターを用いた中置記法になっており、(‘ と ‘) を用いることで入れ子表現も可能である。結合オペレーターとしては CDP データベースのものと同様に、縦に並べることを表現する ‘/’ と、横に並べることを表現する ‘\*’ と、その他を表現する ‘@’ の 3 種類である。この他に、部品の削除と置換を表現するためのオペレーターが存在する。部品の削除は  $A - B$  という式で表現され、これは文字  $A$  から部品  $B$  を削除したものを表現している。例えば、草冠は  $草 - 早$  で表現できる。部品の置換は  $A - B + C$  という式で表現され、これは文字  $A$  中の部品  $B$  を  $C$  に置き換えることを表している。例えば、「花」は  $草 - 早 + 化$  で表現できる。

<sup>12</sup>当時はネットニュース全盛期だった。

この削除と置換を用いることで、3種の結合オペレーターだけでは十分に表現できないような漢字でも、多くの場合において曖昧無く表現可能である。次により複雑な例として((((瞭-目)-小)-日+(工/十))\*支)/皿を示す。CBETA 外字表現は小数の規則で驚く程多くの漢字をカバーできる半面、複数の式表現が生じやすいといえる。

後に、ISO/IEC 10646-1:2000 [5] でも IDS (Ideographic Description Sequence) と呼ばれる前置記法を用いた漢字構造情報記述のための標準形式とそのためオペレーター群である IDC (Ideographic Description Characters) が定義されていたりする。また、「文林」というシステムにおける漢字グリフ合成のための表現形式として、CDL [1] という XML に基づく形式が提案されていたりする。

このように、文字表現指向や字形合成指向、そしてこれらに加え CDP は辞書指向だったり「文林」は教育目的指向だったりして、狙いはいろいろあるものの、部品指向のコード化も根強いものがあるといえよう。ただ、部品指向のコード化も理想の解ではないということが容易に判る。部品指向のコード化がうまく行くのは、部品の数が漢字の数に比べてずっと少ない場合だが、これはあまり自明ではないというか立場によって異なる。部品レベルの異体字をどこまで包摂するのかとか字形合成の際に部品の変形をどこまで機械的にやるのか(それによって、異体部品の包摂の度合が左右されたりする)という漢字の場合と同様の問題がある。そして、漢字の場合と同様、なるべく包摂する立場(用途)となるべく分離したい立場(用途)がある。また、ある特定の漢字でしか使いそうにない部品というものもあり、有限の部品集合で理論上可能な全ての漢字の集合を表現するのは不可能であったりする。オペレーターの種類の問題もある。大概の漢字は左右や上下、垂れや構えといった特定のパターンに分類できるが、交差したり微妙に接触したりといった例外もある。CDL では線や座標とかも書けるようにすることであらゆる漢字(部品)字形を表現可能にしているが、これではもはや漢字専門の SVG [10] の様相を呈しており、漢字の構造のコード化というテーマからは隔たっている。そういう訳で、漢字構造表現だけで漢字の符号化の全てを担うのはちょっと労力に見合わなさそうで、既存の漢字符号化を補完するものとして捉えるのが良さそうである。実際、ここで紹介した各種方式は

そうした位置付けをされているといえる。

### 1.2.3 漢字符号化の現実と憂鬱

そういう訳で、結局の所、1文字づつ文字を同定しては符号化しその包摂する範囲を決めるという JIS なり Unicode なりがやっているやり方が無難だということになる。無論、昔のように文字表を印刷して見せるというだけではだめで、文字を同定するのに必要な情報やヒント、例えば、部首や画数、音訓、用例や包摂規準などが要るし、異体字関係の情報なども要る。しかし、規格化するに当たってこうした情報が必要であるということは、規格を作るのが昔以上に面倒であるということの意味する。それも、単に収録文字数に比例するのではなく、その2乗やそれ以上のオーダーで手間が増え得る。およそ人間の仕事ではない気がするが、符号化しないとコンピューターに乗らないということと、符号化するためにこういう作業をしなければならぬということを鑑みれば、最終的にはどこかで人手による作業が必要となるという結論になる。でもって人間がやる訳であるから間違いは付き物である。また、労力がかかるので文字を増やしたくなくなったり、逆に丸投げして整合性を犠牲にすることになりがちである。また、収録文字数を増やすと文字の性質に関する情報の品質がどんどん下がって行き、結局、符号化文字集合の品質も下がることになる。そして、標準の場合、標準化に付き物の政治闘争も待っている。また、めでたく闘争に打ち勝っても普及するのに時間がかかる。

## 1.3 文字コードの憂鬱

このような訳で、文字コードという仕組(以下では「符号化文字モデル」と呼ぶことにする)の中で頑張ってもどうも理想の世界には辿り着けないらしいということに思い至ったのは1990年台の半ばから後半頃にかけてだっただろうか。

ただそれに代わる方法はすぐには思い付かなかったもので、鬱鬱と漢字のデータベースを作っていたのである。<sup>13 14</sup>

<sup>13</sup>でも、実をいえば、これは結構楽しいのである。ただ、後に述べるようにこの形式もなかなか決定版がなく、暫くすると形式を変えたいのが難点である。

<sup>14</sup>こういう現実逃避をせずちゃんとまっとうな研究をしていれば

## 2 画像を使うこと

符号化文字モデルに代わる文字表現や文字処理について漠然と考えていた頃のある日、研究室に転がっていたどっかの研究会の報告書を何気なく読んでいたら、「ビットマップ計算」というものに気づいた。ここでは図形認識技術に基づき、ビットマップ画像として表現された文字も通常の電子テキストのように操作可能なものにしようとしている。OCR にせよ組版にせよ、文字コードに基づく電子テキストという概念を前提にイメージとの間での変換処理という問題設定を行っている訳だが、考えてみればこれはそれほど自明なことではない。OCR が文字コード列を出力するのは文字コード列でテキストを表現しているからであり、原理的には OCR における特徴ベクトル列でテキストを表現したって構わない訳だ。必要なのは電子テキストとしての利便性、すなわち、検索可能性であるとか編集・再利用の容易さであるとかであって、そうしたことが満たされるならば必ずしもテキストを文字コード列で表現しなくても良いといえる。

このような考えから、文字コード列の代わりに絵でシンボルを表現する Lisp 処理系である “My Symbolic System”<sup>15</sup> というものを考えてみた。<sup>16</sup>

記号という観点からいえば、Lisp は 2 つの世界があれば構成可能であるといえる。すなわち、指すもの（表象）の世界と指されるもの（オブジェクト）の世界である。そしてこの 2 つの世界を symbol が橋渡しする。利用者は指すもの世界だけしか指示できないが、それを評価することにより、指されるもの世界を操作できる訳である。通常の Lisp では symbol の文字列によって表象される。つまり、関数 symbol-name の返り値は文字列となる。利用者は文字列によって symbol を表現するのである。しかしながら symbol を文字列で表現するのが Lisp の本質ではないと考えられる。文字列でなくても、例えば、絵で symbol を表現したって良いのだ。My Symbolic System はこのような観点

道を踏み外さずに済んだのかも時々思うのだが、そもそもある山奥にある情報系の大学院の後期課程にまで行ってしまったのが間違いの元だったのではないかと思わずにはいられない。安価な常時接続のインターネットプロバイダーがもう少し早くに登場していれば良かったのかも。

<sup>15</sup>命名者は g 新部氏である。利用者が自分の作りたいシンボルが作れるシステムという意味が込められている。また、当時、g 新部氏は “My ...” という名前に凝っていたような気がする。

<sup>16</sup>別に Lisp でなくても良いのだが、著者が Lisp 好きなのと、GNU Emacs との比較の点でとりあえず Lisp で考えてみたのである。また、当時は実際に実装するつもりだったということもある。

に基づき構成された Lisp 処理系である。

My Symbolic System は起動すると graphic editor のような画面が出て来る。その画面の中には、car, cdr, cons, lambda, apply などといった Lisp の組み込み関数に相当する icon が見える。また、小さな canvas も見える。canvas には絵を書くことができる。canvas の枠の外には確定用のボタンもあり、それを押すと canvas に書いた絵が icon となって全体の大きな画面に現れる。そう、その小さな canvas こそが intern<sup>17</sup> なのである。

My Symbolic System には文字はないので、もし文字を使いたいなら必要とする文字を My Symbolic System の組み込み機能を使って定義する必要がある。すなわち、生の My Symbolic System には ASCII 文字もないので、ASCII 文字を定義しない限り、ASCII 文字を使って Lisp の program を読み書きすることもできない訳である。

My Symbolic System を考えていた頃、結構真剣にこのようなものを実装するつもりでいた。しかし、実際には “My Symbolic System” のアイデアの内、とりあえず「文字を定義する」という部分を具体化すべく “UTF-2000<sup>18</sup> Project” を始めた。

ただ、画像に基づくテキスト表現に対する未練はその後も続いていて、SVG [10] などを用いたテキスト画像のマークアップなどを細々とやっていたりする<sup>19</sup>し、いつかまとまった時間ができたら My Symbolic System のような画像に基礎を置いた Emacs のようなものを作りたいとは思っているのだが...

## 3 集合を使うこと

UTF-2000 Project で狙ったことは、「文字を定義すること」「定義した文字を使うこと」を比較的少ない労力で現実的に実現することである。このことは言い替えば、現在の符号化文字モデルに基づく実装を比較的少ない労力で置き換え可能な新しい文字処理モデルを作ろうということである。

<sup>17</sup>シンボルを生成する関数

<sup>18</sup>これまた g 新部氏の命名である。「UTF-7 とか UTF-8 とか UTF-16 では弱い。とりあえずもっと景気良く大きくしよう。もっと長い文字表現を使ってみよう。いっそ 2000 bit ぐらい」という意味で付けたという説と、当時の守岡の任期の終わりを表すという説がある。

<sup>19</sup>安岡氏も PDF や DjVu を用いた試みを行っている。[12]

「文字を定義するということ」や「定義した文字を使うこと」は文字をオブジェクト指向的に扱うことであると考えられる。符号化文字モデルでは文字を文字コードでの符号位置という整数値で表現し、このような整数値による文字の表現が隠蔽されていない。それ故、文字を扱う場合、抽象的な文字ではなくて文字コードを意識する必要があり、文字コードに重大な意味が付与される。著者は先ずこれを止めようと考えた。

このように文字をオブジェクトとして扱いその内部表現を隠蔽するのは良いとして、問題となるのは、その文字をどのようなものとして捉えどのように表現するかである。文字という概念の多義性や多用途性、異体字問題をはじめとする漢字の同定の困難さや文字間の関係の複雑さを考えれば、この問題は文字に関する複雑な意味ネットワークの開発に行きつきそうである。しかし、UTF-2000 Project では完全さよりも実現可能性を優先することにしたので、比較的簡単でそこそこの表現力を持った表現方法が望ましい。そこで採用したのが、文字を性質の集合で表現する方法である。これを『Chaon モデル』と呼ぶ。<sup>20</sup> また、ここで表現対象となる「文字の性質」を『文字素性』(character feature)<sup>21</sup> と呼ぶ。文字素性は名前と値の対であり、文字素性の集合は Lisp 的にいえば連想リストに相当する。そして、この連想リストを『文字指定 (char-spec)』形式と呼ぶ。

符号化文字モデルに基づく文字処理システムは固定長ないしは可変長の文字列を用いているが、ここで文字コードの代わりに文字オブジェクトの ID を入れてやり、また、処理の種類に応じて文字素性の参照を行うように修正すれば<sup>22</sup> 原理的には Chaon モデルに基づくシステムに置き換えることができるはずである。また、符号化文字モデルに基づく文字処理でも符号位置をキーにした表を引くことは結構あるはずで、計算量的には大差ないと考えられる。そして、このことを実証するために XEmacs Mule に Chaon モデル的文字処理を導入した XEmacs UTF-2000<sup>23</sup> の開発を行った。

XEmacs UTF-2000 は文字指定形式で表現された文字素性の集合を引数にとり、文字オブジェクトを定義する関数 `define-char` を持っており、Lisp プログラムの形で文字を定義することができた。また、文字素性

を操作するための関数を追加した。Mule-charset は文字素性的一种として再定義され、Mule 機能を Chaon モデル的に再解釈し適切に位置付け直す作業を行った。

## 4 包摂の目を細かくすること

Chaon モデルそのものは文字を素性の集合で表現したオブジェクトとし、文字素性で文字を扱うためのインターフェースを提供する一方、文字オブジェクトの内部表現は隠蔽するという単純なものであり、文字や文字素性に関してほとんどなんにも言っていないに等しい。このことは、こうしたものの意味が文字データベースによって定義されるということの意味している。このことは利用者が望ましい意味付けができるということでもあり、文字素性の集合で表現された文字オブジェクトが自分自身の意味を知っている(というところちょっと大きさが)あるいは文字オブジェクトには文字素性の集合で表現された文字オントロジが付随するといえる。

文字データベースの構成法はいろいろ考えられる訳だが、例えば Unicode のデータベースを用いれば Unicode のセマンティクスを採用することになる。網羅性の点を考慮すれば単純にそうすれば良かったのだが、当時はやりの「Unicode の漢字統合はけしからん。とにかく分ける。必要なら混ぜれるはずだ」という説に従い、<sup>24</sup> 非漢字に関しては Unicode データベースに従ったものの、<sup>25</sup> 漢字に関してはかなり細かく文字定義を分離した。

分離できるということは逆にいえば統合できるということであり、XEmacs UTF-2000 では文字定義を適切に統合し、文字オブジェクトが増えすぎないようにするための細工があった。純粋な Chaon モデル的にいえば、文字の性質は利用者が記述したいように記述されるべきで、いかに似た文字に見えようと、記述が異なれば異なる文脈に置かれた文字として区別されて然るべきであるようにも思えるが、メモリの都合や、それ以上に文字データベースのメンテナンス上の都合から、別個に定義された文字指定を条件に従って1つの文字オブジェクトに統合する必要があった。そのため

<sup>20</sup>当初は『UTF-2000 モデル』と呼んでいた。

<sup>21</sup>当初は文字属性と呼んでいた。

<sup>22</sup>と簡単にいっているが、実はこの部分は結構厄介である。

<sup>23</sup>現在は、XEmacs CHISE と改名している。

<sup>24</sup>個人的には割とこのことを信じてなかったもので、ある意味このだめさ具合を見たり、落としどころを見たりしたいと思ってこんなことをしたのだが...

<sup>25</sup>固定長環境でのいわゆる『全角』『半角』に関して、曖昧性がある部分に関しては、非漢字であっても分離を行っている。

に用いたのが Mule-charset に関する文字素性 (CCS 素性) である。もし CCS 素性が同じなら、同じものと看做す訳である。ただ、符号化文字集合への対応関係は書きたいが、同じものと看做してまとめられるのは困る場合もあり、後に、CCS 素性は双方向写像のものと片方向写像のものに分けられた。<sup>26</sup>

文字定義が分離されている場合に対応関係が判らないと不便であるので、そのための工夫も採り入れられた。Chaon モデル的にいえば集合演算で判断することも原理的には不可能ではないが効率が悪いので、単一の文字素性で表現する手法を用いることにした。そこで用いたのが対応する UCS の符号位置を表す =>ucs<sup>27</sup> と大漢和番号であった。いかにも Chaon モデル的でないのだが、Chaon モデルを考えたはずの肝心の著者の頭の中が Chaon モデル的になっていないという問題があり、また、以前から細々と作って来たデータベースの 1 つが大漢和番号ベースになっていたのでこのような選択を行ったのだった。

このように曲がりなりにも包摂の目をいじる環境を整えて文字データベースをいじりはじめた訳だが、人間がやる以上、どうしてもミスが生じがちである。もちろん、形式上のミスやクリティカルな間違いは define-char 形式のデータを Lisp 関数として評価した時に機械的に判定できるのだが。<sup>28</sup> しかし、これだけでは不十分なので、コンパイラのチェックと同様な機械的な検査を行うことにした。これは、

1. 文字データベースを書き換える
2. 更新された文字データベースを用いて XEmacs UTF-2000 を再コンパイルする
3. 作成した XEmacs UTF-2000 を用いて、文字データベースを出力する
4. 出力した文字データベースと再コンパイル時に用いた文字データベースを比較する
5. 両者に差がなければ検査終了

<sup>26</sup>前者は =ucs のように表記され、後者は =>ucs のように表記されるようになった。

<sup>27</sup>当初は ->ucs と書いていた。後に、->foo は文字間の関係を表すものとして用いるようになったため、=>ucs に変えた。

<sup>28</sup>いじった文字定義を使って XEmacs UTF-2000 を再コンパイルする訳である。すると、syntax error や、文字定義読み込み後の Emacs Lisp のプログラムの load 時や評価時の error で文字定義のミスが検出できる。

6. 差があれば、両者を比較検討し、マージ結果を文字データベースに反映し 2. へ戻る

というような手続きである。機械的といったものの、差異を検討し、マージする作業は人手でやる訳であるから、半機械的というのが正しい。

ここで問題となるのは 2 つの文字データベースの比較であるが、文字定義の正規形と文字定義のソーティングルールを決めて、diff で比較するという安直な方法を使った。このため、文字のソーティングルールの (多分に趣味的な) 試行錯誤を繰り返すことになる。

文字定義の書き換えは、無論、手でいじることができる訳だが、対象となる文字数が数万~十数万以上であることを考えれば 1 つ 1 つ手でいじるのはあまり現実的ではない。そうすると、既存のマッピングデータなどを元に機械的にデータを取り込み、2. へ進むのが現実的である。ここで、Unicode セマンティクスを採用していれば、6. の差異の検討で Unicode ベースでの機械的統合ができるのでとても作業が楽なのだが、細かく文字定義を分離するという方針を立ててしまった以上、人手で見に行くしかない。<sup>29</sup> また、たとえ Unicode セマンティクスを採用していても、マッピングデータに誤りがある可能性があれば、やっぱり人手で見に行くしかない。そして、マッピングデータには誤りが付き物なのだ。誤りとはいえないまでもデータ作成基準の差というのもある。そういう訳で、この部分は割と地獄の作業となる。

## 5 いいかげんな定義を使うこと

このように地獄の作業は進まず、下手すると死ぬまでひきこもりかねない危険な作業である。<sup>30</sup> 肩も凝るし健康にも悪い。いろんな誤りを発見したりして、1 文字のために辞書や規格書を引き回ったり、挙げ句の果てに辞書や規格書の誤りも見付けたりで、「やっぱり他人が作ったデータは信用できない。自分の目でチェックしなければ」などと疑心暗鬼になり、ますますど壺に嵌まってしまうのである。いずれにせよ絶えず中途半端なデータを使わざるを得ないことになる。

<sup>29</sup>ある程度その方針で進めようと、なかなか引き返せないものである。

<sup>30</sup>この過程で、著者が書いた幾つかの Emacs Lisp パッケージの幽霊メンテナー化が進行し、今では mailing list に記事を書くのも勇気がいる有り様である。

ただ、実の所、これでも実害がなかったりする。というのも、読み書きする電子テキストというのは文字コードで符号化されていて、その範囲で round-trip conversion ができさえすれば（そして文字が見えさえすれば）あんまり困らなかつたりするものなのだ。<sup>31</sup>

あんまり困らないのは幸いなことだが、では一体全体なんのために文字定義を作ってるのかと自問自答せざるを得ないことになる。これはそもそも文字とか文字素性というものが何かということを書き上げて、文字データベースを作り始めた結果、趣味的に文字データベースを作ることが目的となって本末転倒してしまった結果といえる。また、Chaon モデル的なアプリケーションの少なさの問題でもある。それは真の意味での XEmacs UTF-2000 アプリケーションの少なさという問題でもあるし、Chaon 的世界が XEmacs UTF-2000 に閉じているという問題でもあった。

## 6 CHISE Project のこと

XEmacs UTF-2000 によって、曲がりなりにも実現した Chaon モデル的文字処理であるが、それを意味のあるものにするにはやはり文書処理環境全体を Chaon モデル的文字処理に変えて行く必要があるように思われた。これは言うは易し行うは難しで、私一人でやるには限界があり、仲間を集めることが重要であった。

XEmacs UTF-2000 での経験により、アーキテクチャの大まかな方針は定まっていた。それは、文字オントロジーを蓄積した文字データベースを中核として、それを参照して文字処理を行うクライアント群を作ることである。

2001 年度には Christian Wittern 氏と共に IPA の未踏ソフトウェア創造事業に採択され、私は XEmacs UTF-2000 の文字データベースの外部化、Wittern 氏は XML TopicMaps 形式での文字データの記述の研究を行った。また、データ面では IDS 形式による漢字構造情報のデータベース化をアルバイトを使って行うとともに、XEmacs UTF-2000 での表現形式や関連ツールの整備を進めた。TopicMaps 形式で文字データを扱うデータベースサーバーの方は、そもそも XML TopicMaps 自体が規格化途上だったり、そのため実装もあんまりなかつたりで、XML TopicMaps 処理系自

体の開発になってしまい、その規格の複雑さもあり、大量の文字データを現実的に扱うことが可能な処理系の実現には至らなかったが、XEmacs UTF-2000 の文字データベースの外部化は実現できた。また、7万字以上を収録した漢字構造情報のデータベース化により、文字の検索やグリフの自動合成などの応用への道が開けた。

2002 年度にはさらに仲間を増やした。大学院時代の友人である鈴木泰博氏の紹介でネットワークの数理解析を研究している藤原義久氏とメディアアーティストの江渡氏に出会い、漢字間の関係の可視化プロジェクトが立ち上がった。また、師氏を仲間に引き込むとともに、氏の紹介で漢字グリフ合成システムを研究している上地宏一氏も引き込んだ。チベット仏教の研究者で GNU Emacs のチベット語サポートの作者にして TeX の多言語拡張 Ω に詳しい苦米地等流氏もお誘いし、その御紹介で宮崎泉氏も参加して頂いた。これにより、Chaon モデルの多面的な展開のための体制が出来上がり、

libchise 文字データベース操作のための基本機能を提供するためのライブラリ

XEmacs CHISE XEmacs [11] に基づく Chaon 実装 ( Emacs Lisp 処理系および対話型編集環境 )

Ruby/CHISE Ruby [9] に基づく Chaon 実装 ( スクリプト言語 ) ... 可視化に関するさまざまな関連プログラムも実現されている

Perl/CHISE Perl に基づく Chaon 実装 ( スクリプト言語 )

/CHISE 多言語 TeX 処理系 Ω [8] に基づく Chaon 実装 ( 組版システム )

Kage 漢字グリフ合成システム [13]

などの開発が進んだ。

## 7 文字間関係を書くこと

実現可能性を考慮して、Chaon モデルでは文字を文字素性の集合として表現することにしたが、やはり文字間関係を記述することは必要不可欠なことである。そこで、=>ucs や大漢和番号のようなものを

<sup>31</sup>無論、困る時は文字定義を直す訳である。

その代用品にしてきた訳であるが、やはり、ちゃんと文字間の関係を書きたくなり、そのための形式を定義した。

Chaon モデル的文字指定形式の枠内で文字間の関係を書くということは、文字間の関係を示す文字素性名を決めて、その素性値の形式を決めるということである。素性値で表したいのは(1つ以上の)文字であろうから、それは文字を表現するもののリストということになる。Chaon モデルの世界では文字に唯一固有のIDを振らない(ないしは、内部的に振っても外に出さない)のであるから、文字を表現するのは Chaon モデル的な文字素性の集合で文字を表現する文字指定形式ということになる。

これでとりあえず文字間の関係は書ける訳であるが、汎用的な大規模文字データベースを目指すのであればちょっと物足りなくなる。というのも、用途や立場・学説などによって、文字素性に値に複数の選択肢を設けたい場合があるからである。もちろん、用途や出典などによって文字素性名を変えれば良いのであるが、アドホックな名前を付けてると訳が判なくなったり、名前を考えるのが面倒になって来たり、大部分共通してる場合に重複する文字素性を付けるのが気が引けたり、でもしようがないからコピーしたり、しなかったりと、スパゲッティプログラムの様相を生じてしまうのである。そういう訳で、きちんと構造化してアドホックな方法を追放することにした。<sup>32</sup>

こういう場合、必要なのは選択肢が書けることと、各選択肢にメタデータが付けられることであるといえる。文字素性値がリストであれば複数の選択肢は書ける訳であるから、必要なのはメタデータの記法であるという訳で、S 式における属性リストを使って<sup>33</sup>『文字参照(char-ref)』と呼ぶ形式を定義した。この形式では、メタデータの種類は属性名で表され属性値はその値を表す。但し、属性名が :char の場合の属性値はメタデータが付加される文字を表す。なお、現在、意味が定義されている属性名としては :sources があり、この値は出典情報を表す。

しかし、CHISE Project がはじまり、文字データベースを C や Perl や Ruby でも利用できるようになる

<sup>32</sup>本当は、Wittern さんが TopicMaps でかっこいい解決をしてくれて、その意味ネットワークと Chaon 的世界の橋渡しをする方法を後で考えたいと思ってたのだが、一向にできる様子がない状況のまま Wittern さんも(私も)忙しくなって現在に至る。

<sup>33</sup>文字指定形式は連想リストなので、属性リストなら区別が付く。

と、このような S 式で構造を表現する方法は不便であり、素性値ではなく素性名を構造化する方法に移行することにした。これは対象となる文字素性の名前(文字素性基底名)に、『ドメイン識別子』と呼ぶ値を選択するための識別子を付けた文字列生成しそれを名前(文字素性具象名)として用いたり、同様にメタデータ識別子を付けた文字列を生成しそれを名前(文字素性メタデータ名)として用いる方法である。この名前は次のような規則で生成される：

文字素性具象名

:= 文字素性基底名 @ ドメイン識別子  
| 文字素性具象名 / ドメイン識別子

文字素性メタデータ名

:= 文字素性具象名 \* メタデータ識別子

例えば、総画数を表す文字素性名を total-strokes とし、ドメイン識別子として ucs を用いる時、文字素性具象名は total-strokes@ucs となる。また、出典情報を表すメタデータ識別子を sources とする時、total-strokes@ucs の出典情報は total-strokes@ucs\*sources で表される。

部首と部首内画数のように異なる種類の文字素性の値が対応関係を持っている場合、ドメイン識別子を用いてその対応関係を表すことができる。例えば、部首を ideographic-radical、部首内画数を ideographic-strokes で表す時、

ideographic-radical@ucs  
ideographic-strokes@ucs

の両者は対応する。

値を構造化する手法と名前を構造化する手法を比べた場合、前者はドメイン識別子を必要としないという利点を持っているものの、値が構造データとなるので C のような単純な記憶管理機構しかない環境では不便である。また、高速化を要するような単純な処理の場合、大抵、複数の値やメタデータを必要としないといえる。また、Chaon モデル的には文字が文字素性の単純な集合になっている方が自然であり便利であるが、値を構造化すると複数の値同士の集合演算を要し、処理が複雑になる。このようなことを鑑み、現在の CHISE では共有文字データベース内では原則として値ではなく名前を構造化する方針を採っている。

このように文字間の関係を書くことが行われ、実際に常用漢字表にある常用漢字と『旧字』の関係のデータをはじめとする幾つかのデータが取り込まれ、実質的に CHISE の文字データベースはネットワーク構造を採るようになった。

## 8 包摂の目をいじること

Chaon モデルを作る時に前提とした文字観のひとつに「文字は状況依存な存在である」というものがあった。Chaon という名前は状況意味論（のインフォ年代数）における infon から採っており、<sup>34</sup> 文字素性は infon、文字素性の集合である文字オブジェクトは infon の 結合であると考えられる。

このように Chaon モデルを作った時の気持としては、観念上・抽象的な文字も具象の・書かれた文字も同様に扱えるようにしたいと思っていたし、文脈依存な・一時的な文字を積極的にサポートしたいと考えていた。しかしながら、現実の XEmacs CHISE での文字オブジェクトは多分に旧来の符号化文字をひきずった観念上・抽象的な文字に偏っていたといわざるを得ない。これはシステムの不十分さやアプリケーションの問題もあるだろうし、利用者としての著者の頭の硬さが原因ともいえるが、いずれにせよ、現状は極めて矮小な実現であるといわざるを得ない。

こうした問題を解決するためには文字オブジェクトが表す文字の包摂の目（文字オブジェクトが表す範囲）が伸び縮みするように感じられることではないかと考えている。これは、伸び縮みさせた文字オブジェクトが気軽に作れたり使えたりすることの必要性や、そういうビューで文字データベースを可視化するユーザーインターフェースの必要性を意味しているといえる。また、文字データベースにおける文字定義を文字オブジェクトというものとして捉えるのではなく、現在編集しているテキスト上の文字っぽいものを解釈するためのとっかかり程度に位置付け、文字っぽいものの流動性を高める（そのように利用者に感じさせる）必要があると思われる。

何はともあれ、そのような思いを込めて、文字定義の継承機能を追加した。この機能では既存の文字オブジェクトに対して親や子供を定義することができる。

<sup>34</sup>守岡家の猫の名前ではない。ちなみに、守岡家には現在チャオという猫がいるが気のせいである。

## 9 おわりに やっぱり文字が好き

CHISE Project に至る過程やそこでの試行錯誤を著者の実体験に沿って述べてみた。おそらく、著者と違う世代や Lisp を知らない読者にとっては読みづらいものになったのではないと思われるが、この点の工夫が十分でないことをお許し願いたい。

その代わりに、どういう問題意識を持って CHISE Project に至りその先を進もうとしているのかに重点をおいて説明しようとした。CHISE Project における文字モデルである『Chaon モデル』がどのような意図で設計され、XEmacs CHISE をはじめとする Chaon 実装がどういう問題意識に基づき開発されて来たのかということ、主に文字定義や文字データベースの観点から説明した。

但し、ここで述べているのはあくまで著者の目から見た道筋であり、必ずしも CHISE Project のメンバー全員がこのような観点を持っているとはいえないことをお断りしておく。また、ここで述べた著者の観点は多分に文字好きものとしてのそれであり、工学者としてそのような著者を冷やかな目で見ている著者の存在を感じなくもない。しかしながら、文字好きとして、文字のさまざまな様相を味わい操作しながらテキストを処理する環境を実現しそれを使うことは楽しいことで、やっぱり文字いじりは止められないのである。

## 謝辞

本論文で述べた CHISE プロジェクトの研究の一部は 2000 年度に旧通商産業省工業技術院電子技術総合研究所（現 独立行政法人 産業技術総合研究所）からの受託研究として行われ、2001 年度には情報処理振興事業協会の「未踏ソフトウェア創造事業」の助成を受けた。

また、忙しい中 CHISE プロジェクトに主要メンバーとして御参加頂いた Christian Wittern 氏、江渡浩一郎氏、上地宏一氏、鈴木泰博氏、苫米地等流氏、藤原義久氏、宮崎泉氏に感謝する。また、2001 年度に未踏

<sup>35</sup>状況理論的に考えるならば、親文字オブジェクトにとって子文字オブジェクトの集合は、子文字オブジェクトの集合の 結合と考えられる。

ソフトウェア創造事業のプロジェクトマネージャーとして、その後も、プロジェクト遂行において貴重なご助言と多大な御助力を頂いた g 新部裕氏に感謝する。

また、横田裕思氏やしおさきかずひこ氏をはじめとする UTF-2000 mailing list の参加者に感謝する。また、XEmacs UTF-2000 の開発にあたって貴重なご助言と御助力を頂いた産業技術総合研究所の戸村哲氏、半田剣一氏、錦見美貴子氏、高橋直人氏に感謝する。

## 参考文献

- [1] Tom Bishop and Richard Cook. A specification for CDL — Character Description Language. In *Proceedings of the Glyph and Typesetting Workshop*, Feb 2004.
- [2] CDP 外字. <http://www.sinica.edu.tw/~cdp/zip/font/eudc.zip>.
- [3] 漢字庫. <http://www.sinica.edu.tw/~cdp/zip/hanzi/hanzicd.zip>.
- [4] International Organization for Standardization (ISO). *Information technology - Character code structure and extension techniques*, 1994. ISO/IEC 2022:1994 (= JIS X 0202, 「情報交換用符号の拡張法」).
- [5] International Organization for Standardization (ISO). *Information technology — Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane (BMP)*, March 2000. ISO/IEC 10646-1:2000.
- [6] International Organization for Standardization (ISO). *Information technology — Universal Multiple-Octet Coded Character Set (UCS) – Part 2: Supplementary Planes*, November 2001. ISO/IEC 10646-2:2001.
- [7] Mikiko Nishikimi, Ken'ichi Handa, and Satoru Tomura. Mule: MULtilingual Enhancement to GNU Emacs. In *Proc. INET '93*, pp. GAB-1–GAB-9, 1993.
- [8] The omega typesetting and document processing system. <http://omega.cse.unsw.edu.au:8080/>.
- [9] The object-oriented scripting language Ruby. <http://www.ruby-lang.org/>.
- [10] The World Wide Web Consortium (W3C). *Scalable Vector Graphics (SVG) 1.0 Specification*, September 2001. <http://www.w3.org/TR/SVG/>.
- [11] XEmacs. <http://www.xemacs.org/>.
- [12] 安岡孝一. 外字と異体字. アジア情報学のフロンティア — 全国文献・情報センター人文社会学学術セミナーシリーズ No.10, 全国文献・情報センター人文社会学学術セミナーシリーズ, 第 10 巻, pp. 25–34, 2000.
- [13] 上地宏一. 漢字フォント自動生成サーバ “影 KAGE” の構築 — 文字コードの枠組みを越える次世代漢字処理の提案 —. 漢字文献情報処理研究, Vol. 3, pp. 143–147, 2002.
- [14] 台湾經濟部中央標準局. 中文標準交換碼 (Chinese Standard Interchange Code), May 1992 (民国 81). CNS 11643.