プログラミング言語Prologの拡張としての MY Server ――自然言語処理を目標とする WEB開発環境の開発――

松　村　保　寿

0．はじめに

本論文は、プログラミング言語Prologを用いて自然言語研究または言語教育のためのWEB上の基盤をどのように開発・構築するかというテーマを追求した、サーバ・ソフトウェア構築のための原理を述べたものである。使用したコンパイラは、Daniel Diaz（Daniel.Diaz@univ-paris1.fr）によるGNU-Prolog（http://www.gprolog.org）である。GNU-Prologのコアの要素を拡張し、PrologによってWEBプログラミングを可能にする言語仕様の拡張を行った。

しかしながら、この原理的な議論よりも言語研究や言語教育への応用研究に特化したアプリケーションの開発と言語教育の実践的可能性により関心のある読者は、まずII.の応用アプリケーションの開発についてのいくつかの研究にまず目を通していただくことをお勧めしたい。

Ⅰ．WEB上でアプリケーションを開発することのできる開発環境としての“MY Server”　―その原理―

【問題点】

自然言語処理にふさわしいとされるPrologが開発されて数十年が経過した[注1)]。この間、欧米、とりわけこの言語が生み出されたフランスやフランス語圏カナダ、またそのプログラミング言語を実際の企業業務に利用している北欧（とりわけ北欧の空港管制システム）などを別にすれば、第5世代研究開発[注2)]の基盤のプログラミング言語に据えたはずの当の日本においてこのプログラミング言語は、急速に関心が失われているように見える。その最大の理由は、自然言語の解析に携わる当の言語学者や大学院生自身がこのプログラミング言語に触れる機会が失われている大学の教育カリキュラムそのものに大きな原因があると思われる。このプログラミング言語は、文系の大学教育のカリキュラムに十分含まれていい、極めて文系的な、ある意味で言語哲学的な内容を含むプログラミング言語である。統計的な数値研究ではなく、その概念体系の関係性の構築を基盤にする研究スタイルは、論理構成の普遍原理（ユークリッドの公理に基づく論証やアリストテレスの三段論法に始まる古代ギリシャ、17世紀デカルトやパスカルの機械論、ライプニッツの普遍言語、19世紀末にはアリストテレス以来最大の論理学者と言われているゴットロープ・フレーゲ、20世紀初頭に形式論理に革命をもたらしたラッセルの分析哲学を経て論理実証主義と科学哲学に連なる）ヨーロッパ哲学の一つの系譜を脈々と受け継ぐ。おそらくその系譜は1960年代後半のN.Chomkyや70年代のR.Montagueの意味論（時間と空間に相対化された形式意味論“Formal Philosophy”）に連なる[注3)]。Prologというプログラミング言語は、その思想性において、言語と言語研

究（あるいは意味論研究）の一つの中核をなすプログラミング言語と言っていい。その意味では、大学における、文系の、とりわけ言語の研究に携わる外国語大学の教育プログラムに含まれていいはずのプログラミング言語でもあり、最も文系らしいプログラミング言語であるとも言える。それにも関わらず、この認識は言語と言語研究に携わる学会そのものにおいても全く共有の認識とはなっていない。

もう一つの理由は、2000年以降、インターネットが急速に普及してきた現代、今日的なスタンダードから見て、このプログラミング言語がWEB上のプログラミングを行うには、他のプログラミング言語やスクリプト言語JAVAやPHPなど、またWEB上で動画を扱うFlashスクリプトなどに比べると、プログラミング言語の核であるpure prologのみでは、表面のユーザ・インターフェースやグラフィカルなアピアランス（＝見かけ、見栄え）を作るという部分では、かなり見劣りがするということもまた否めない事実であろう。そのため、PrologはWEB上のプログラミング言語としては不適切ではないかという見方が支配的であると思われる。しかし、私は、Prologは他のプログラミング言語と同等にWEB上のプログラミング言語としても全く遜色なく用いることができると考えている。

【解決への試み】

この最後の問題、すなわちPrologをWEB上のプログラミング言語として拡張するための試みとして、グラフィカルな表面はWEB上の画像（もちろん音声やVideoの使用も可能である）で構成し、実質的なデータ処理や自然言語処理をProlog本来の機能でプログラミングできるように、プログラミング言語の拡張を図る。これによって他の現代的なプログラミング言語（C++、C#、JAVA、あるいはスクリプト言語としてのPerlやPHP）と同様な処理を比較的簡単に行えるようにすることを試みる。それは、コンパイラとしての言語処理系にWEBプログラミングに必要なHTML / XMLを解析プログラム（HTML / XML Parser）を追加の機能（述語）として付加

する。これよって、さまざまなWEB上でのコミュニケーション・ベースのアプリケーション・プログラムを開発することが可能になる。しかもアプリケーション・プログラムを開発するための基盤となっているプログラムそのものがPrologという「人工知能言語（artificial intelligence language）」で書かれているため、この基盤の上に開発されるアプリケーション・プログラムを、自然言語処理や自然言語の構造解析処理（たとえばgenerative grammar）などの言語研究のためのプログラムとして開発することが容易になるはずである。したがって本論文に添付されているプログラム（この全体構想を“MY Server”と呼ぶことにする）については、≪その上でWEBアプリケーションを開発することができる基盤のソフトウェアである≫という意味で、―敢えて大胆な比喩を用いれば―「**WEBのブラウザ上に展開する自然言語研究のための基盤のソフトウェア、すなわち『WEB OS』を実現している**」という位置づけをすることが可能と思われる。（もちろん、本来の意味でのOSとは似てもつかない代物であるということは重々承知の上である。）

このようにして、“MY Server”上では、Prologというプログラミング言語を用いてWEB上に展開するコミュニケーション･プログラムを開発することができる。しかしながら、テスト問題の自動採点というアプリケーション群に関しては、決してプログラミングを必要とするものではない。むしろWEB上の基本テキストHtmlの知識とWEB OSとしてのMY Serverの持つ、アクセス・パーミッション（ユーザ・アクセスの制限）を巧みに組み合わせれば、Htmlの知識のみで、一個の教育教材アプリケーションを開発できるのである。この試みについては、II.部のアプリケーションの開発の事例を参照されたい。

しかし、“MY Server”が目指している目標は、必ずしも有用性／実用可能性の研究ではない。むしろ、自然言語の研究のための言語学者のためのツールとしての自然言語解析プログラムである。たとえば、WEB上で、ユーザから返ってくる文字列、あるいは自然言語のテキストをProlog固有

の自然言語処理プログラムや自然言語の構造解析プログラムに載せることを可能にする。目指す遠大な目標は、ユーザから返って来たテキストの自動処理である。この最後の点にこそ、本論文を敢えて名古屋外国語大学の言語研究の一つとして発表する所以である。

なお、この試みは、世界のProlog開発／研究者の集まりである集会（Faro ポルトガル2006, Sankt Peterburg ロシア 2008）で発表され、Prologでサーバを構築するという試みは、世界的にも非常に珍しい試みとして（おそらく世界的にも他に例がないだろうと思われる）と認められ、その後フランスのPrologコンパイラの開発元のWEBサイト（http://www.gprolog.org）に採用された（2010年以降）研究でもある。

以下の記述は、以上の開発コンセプトの概念的な説明である。ただ、残念ながら紙数の制限のため、大雑把な全体構想を述べることができるのに過ぎない。またプログラムのソースコードもこの基本アイデアを示す部分のみしか本論文には添付していない。全ソースコードは、名古屋外国語大学のWEBサイト（http://www3.nufs.ac.jp/~matsmura/）からダウンロードできる。今後も随時拡張を続けていく予定である。

1．WEB OS としての“MY Server”

Prolog Html という概念（松村）
添付プログラム：prolog_html.pl

Prolog Htmlとは、次のようなリスト構造と定義する。
リストの各要素を次の6つのもののいずれかとする。

i) 一般のタグ構造 （開始タグと終了タグを持つタグ）をt/3構造で表す。
t(Tag, AttributeList, TagBlock).

例：

```
t(table,[border=1],[
        t(tr,[],[
                t(td,[],[
                        'テーブル・セル'
                ])
        ])
])
```

これはHtmlにおける次のようなタグ構造に相当する。

```
<table border="1">
        <tr>
                <td>
                        テーブル・セル
                </td>
        </tr>
</table>
```

タグの名前 Tag：

アトム

例：table, tr, td など

属性リスト AttributeList：

等号による構造（属性）または文字列を要素とするリスト

例：[border=1]など。場合によっては固有属性を用いた[name=select, multiple]のような属性リストもあり得るだろう。

タグ・ブロック TagBlock:

開始タグ（opening tag）から終了タグ（closing tag）までの間の部分。これはまた帰納的にProlog Htmlとなっている。

なお、Prolog Htmlにおけるタグ構造をPrologの構造として表記するときには、以下では習慣として（したがって、絶対的な規約ではないが）t/3構造の第3要素のリストを書き始める最初の鍵括弧 [の直後で、改行記号を入れるようにする。そしてその下位の行においては、tという関数子の直下でこの鍵括弧に対応する、もうひとつの鍵括弧] とt-構造に対応する丸かっこ) を連続して書くことにする。

例：

```
t(table,[border=1],[      % t-構造の第3要素の [ の直後で改行。
    …
    …
])                        % ] で、第3要素のリストを閉じ、
                          % かつそのの直後でt-構造を ) で閉じる。
```

ii) 空タグ（終了タグのない、いわゆるempty tag）をe/2構造で表す。

例： e(img, [src='/wwwroot/images/image.png'])

これは<img src="/wwwroot/images/image.png">という空タグのProlog表現である。

iii) XML形式の空タグをx/2構造で表す。

例： x(img, [src='/wwwroot/images/image.png'])

これは というXMLの空タグのProlog表現である。

iv) スクリプト言語の挿入をq/2構造で表す。

例： q(php, [echo, 'date(¥"Y/m/d\")'])

これは <?php echo date("Y/m/d"); ?>というスクリプト言語の挿入を表すProlog表現である。

v)　コメントをc/1構造で表す。

例：　c(‘これはコメントです。’)

これは <!-- これはコメントです。--> というコメントの挿入を表すProlog表現である。

vi)　文字列は、Prologのアトムとして表現する。

例：　次のt/3構造に含まれる1重引用符に囲まれたアトム‘みなさんこんにちは。’　が、Prolog Htmlにおける文字列である。

```
t(td,[],[
        ‘みなさんこんにちは。’
]
```

例：次のようなHtmlは、その下に書くようなリストとして表現される。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD html 4.01 Transitional//EN">
<html>
    <head>
        <title>
            みなさん、こんにちは
        </title>
    </head>
    <body style=”background-color:yellow”>
        世界のみなさん、こんにちは。
        <br>
        <img src=”image.png”>
    </body>
```

</html>

以上の構造化によってすべてのHtml/XML表現がPrologのリストとして表現できる。以下はその一例である。

```
[
        c('DOCTYPE HTML PUBLIC \"-//W3C//DTD html 4.01 Transitional//
        EN\"'),
        t(html,[],[
                t(head,[],[
                        t(title,[],[
                                'みなさん、こんにちは'
                        ])
                ]),
                t(body,['background-color'=yellow],[
                        '世界のみなさん、こんにちは。'
                        e(br,[]),
                        e(img,[src='image.png'])
                ])
        ])
]
```

本論文で提供する最初のプログラムは、このようなHtml / XML言語で書かれたソース・ファイルをPrologのリストに変換するプログラムである。それはprolog_html.plという715行程度のPrologプログラムであるが、ただ本論文の性格上、このPrologソースコードそのものは、本論文に添付される付属資料として論文CD-ROMに焼きこんでいただくことにする。この紙面上では、ただ概念的な説明ができるのみである。

Prolog処理系（Prolog system）を基盤にしてHtmlファイルを自動加工させてサーバ・システムを構築する場合には、もちろん元のHtmlテキストを常に解析（パースparse）するのではなく、本論文で提供されているプログラムprolog_html.plを用いて、事前にHtml言語で書かれたテキストをPrologのリストに変換して得られるProlog Htmlのソース・コードをサーバの基本データとして用いる。したがって、このWEBサーバは、Htmlテキストを基礎データとしているものではなく、Prolog HtmlというPrologのソース・コードを基本のデータとして持つ、純粋に「PrologによるWEBサーバ」を構築することになる。多くのサーバを構成する基本データのうち唯一の例外である、トップページindex.htmlを除いて。

1．HtmlテキストをProlog Htmlに変換するプログラム：prolog_html.pl

prolog_html.plを構成しているプログラムは、Htmlテキストを一文字一文字読みながら、Prologのリストを作り上げるプログラムread_html.plと、それによってメモリにロードされたProlog HtmlをTABによるインデントを付けて“pretty-print”された形で書き出すwrite_phtml.plとである。後者のファイル名には_phtmlという文字列が添えられているが、これは「Prolog Htmlとして書き出す」という意味をファイル名で暗示するためである。

また逆に、prolog_html.plは、拡張子が.phtmlとなっているProlog Htmlファイルを一般のHtml形式で書きだすこともできるようになっている。

2．プログラム：read_html.pl

read_html.plは、Htmlテキスト・ファイルを一文字一文字読み込みながら、’<’（Asciiコード番号60）を読み込めば、「タグの文字列の読み込み状態」とし、またその状態で’>’（Asciiコード番号62）を読み込めば、「Htmlの文字列を読み込んでいる状態」として、状態を切り替えながらファイルを読み続ける。最終的にファイルのおしまいに達すれば、読み込みを終える。

タグの文字列の読み込み状態ではない、一般のHtmlの文字列の読み込み状態のときには、それ以上の解析処理プロセスは不要であるが、「タグの文字列を読みだす」状態のときには、かなり複雑な処理が介在する。

「タグの文字列を読み込む」状態のときには、また最初に読み込まれたタグの名前を切り出し、そのタグの名前について

i) 最初の文字が/であるか、否かを判断する。
最初の文字が/であれば、これは終了タグ（closing tag）であると判定する。
例： </table>のようなケースである。

ii) 最初の文字が!であるか、否かを判断する。
最初の文字が!であれば、これはコメントであると判定する。
例： このコメントには、<!-- -->のように一般のコメントの場合もあれば、また<!DOCTYPE …>のようなHtmlファイルの冒頭の宣言をするケースもある。

iii) タグの文字列の最後の文字が/であるか否かを判断する。
これはXML形式の空タグであると判断する。
例：
のようなタグである。

iv) 最初の文字が?であるか、否かを判断する。
最初の文字が?であれば、これはスクリプトを挿入するタグであると判定する。
例： <?php … ?>のようなケースである。

以上のいずれでもなければ、

v) これはこのタグは、空タグ（closing tag）と判定する。

ただし、上記のような判断ができる前提は、「タグの文字列をまずスペースで区切って、タグの文字列の最初の文字列を切り出すことができなくてはならない。このスペースで区切るという処理も、実際にはもっと複雑な

プロセスで、半角スペース、全角スペース、TAB、改行記号などいわゆる「ホワイト・スペース（white-space）」のすべてを考慮しなければならない。さらに、タグの文字列として鍵括弧<と>との間に含まれるものには、Htmlには属性が含まれるはずである。それは等号'='で結ばれた二つの文字列である。その右辺はしばしば2重引用符で囲まれている（場合によっては1重引用符のこともある）。これらのことをすべて考慮しながら、「タグの文字列をまずホワイト・スペースや等号、引用符などで区切る」処理プロセスを以上のi)からv)までの判断に先行させなければならない。しかしこの部分の処理プロセスには触れないでおく。

３．終了タグから遡って開始タグまでを検索

以上のプロセスでは、本来のタグ、すなわち開始タグ（opening tag）と終了タグ（closing tag）とを共に持ち、両者のタグの間にまたHtmlの構造を再帰的（recursive）に持っている構造をまだ切り出すことができていない。v)によって、すべてのタグは当初、空タグ（empty tag）のe/2構造として認定されているに過ぎない。したがってこのe/2構造から、t/3構造を生成する処理を受け持つアルゴリズムを組み込まなければならない。

Prologのリスト構造という性質のため、文字列データは、空のリストをスタートの状態として、常にそのリストの先頭にデータを読み込んでくる。したがってデータは元のファイルに書かれている順序とは、逆順に読み込まれていることになる。これはデータ構造の分析のためには、ある意味で都合がよい。というのは、終了タグ（closing tag）を読み込んだ瞬間に、それまで読み込まれていたタグのリストを逆順に遡って、対応する開始タグ（opening tag）を探索すればよいからである。

ある時点で、終了タグ（この構造を暫定的にct/1構造としよう）が読み込まれると、そのタグ名と同じ名前を持つ空タグを、メモリに蓄積されたリストで逆順で探索する。リストそのものには、逆順に蓄積されているので、実際には、リストの冒頭から検索することになる。もしも同名の空タ

グが最初に見つかれば、それが対応する開始タグ（opening tag）であるとして認定してやればよいことになる。

概念的には、次のような処理プロセスを構成することになる。なお、ここにct/1は、closing tag（終了タグ）のことである。

```
ct(Tag)
    …
    ….
e(Tag, AttributeList)
```

したがって、同名の空タグ（e/2構造）が見つかったときには、これら全体を逆転させて、開始タグと終了タグを両者とも兼ね備えた、かつその間のタグ・ブロックと称するHtml構造を持つt/3構造の切り出しに成功したことになる。

```
t(Tag,AttributeList,[…])
```

こうして、ct/1構造とe/2構造の少なくとも2つのもの（多く場合、その間にある下位のHtml構造が介在しているため、2個以上の構造）が、t/3構造という一つの構造に集約されていく。こうして、このプロセスを帰納的（recursive）に繰り返すことが可能になる。以上のプロセスを繰り返し、最終的に終了タグ（closing tag）が無くなった時点で処理全体の終了とすればよいわけである。この間、終了タグに対応する開始タグが見つからないケースが仮にあったとすれば、それは元のHtmlファイルそのものが持つエラーであり（不正なHtmlファイル）、この場合には当該の終了タグを無視して処理を続け、結果としては不整合なProlog Htmlが生成されることになる。しかし、この例外処理によって、処理プロセスそのものが失敗するわけではない。

以上がread_html.plを構成するプログラムの概念的な解説である。このread_html.plは、prolog_html.plを構成する一番の核になるプログラムである。ソースコードそのものは、本論文集のCD-ROMに添付のファイルを参照していただきたい。

４．ソースコードを書き出すプログラム

write_phtml.plとwrite_html.plがメモリにロードされたリストとしてのProlog Htmlを書き出すプログラムである。これらのプログラムは、比較的簡単なプログラム（100行ないし150行足らずのソースプログラム）であるため、特に解説なしで、そのソースを本論文のCD-ROM直接添えておくことにする。コアの考えは、単にタグや文字列のひとまとまりの部分には、TAB文字（Asciiコード9番）によってインデントを付けながら改行する。それによってリストをいわゆるpretty-printするプログラムである。

５．CGIプログラムとしてサーバを構築する。

端末から返ってくる情報を受け取るプログラム：get_name_value_list.pl

WEB上でインタラクティヴに情報をやり取りするためには、まずHtmlソースとして、ユーザに書き込み欄を提供する必要がある。たとえば、このPrologサーバ・システムでは、次のようなProlog Htmlを用意することになる。

```
[
        t(html,[],[
                t(body,[],[
                        e(input,[type=text, name=input_text,value=''']),
                        e(br,[]),
                        e(input,[type=submit,name=submit,value='Send']),
                ])
```

```
        ])
]
```

このProlog Htmlをプログラム write_html.plを使って書き出すと、端末ユーザのブラウザ上に次のような画面が表示される。

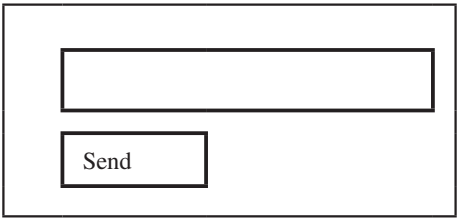

図1

このブラウザ画面において、ユーザが何か（たとえば：Hello!）を書き込むとその答えは、サーバ側に次のような文字列として返ってくる。

```
input_text=Hello!&submit=Send
```

すなわち、ユーザからサーバに返ってくる情報は、ユーザが見た画面情報の全体ではなくて、次のような極く制限された情報がサーバに返ってくるのみである。すなわち、サーバが送った元のHtmlに含まれるある特定のタグ（上の例では、inputタグ）に含まれる属性name=input_textの右辺の値input_textとユーザが入力した値を右辺にもつvalue 属性（value=ユーザが入力した値）の右辺とのペアを等号で結んだ形式

```
input_text=Hello!
```

と、もう一つのinputタグのname属性とvalue属性のそれぞれの右辺を等号

で結んだ形式

submit=Send

とがアンパサンド（&）で結ばれて返ってくるのみである。

この文字列をリストとして捉えるプログラムが、get_name_value_list.pl である。

このプログラムは、結果として次のような構造（等号'=' を用いた構造）のリストを返すようになっている。Get_name_value_list.pl を gprolog インタープリターの上で実行すると

```
?- get_name_value_list(NameValueList).
NameValueList = [input_text='Hello!', submit='Send'].
```

実際はWEB上の標準ストリームから情報を受け取って、それをこの最後のリストとして返してくる。

6．ユーザからの情報をサーバ側で処理して、新しい画面を返す

プログラム：　reconst_html.pl

第5節で述べたように、ユーザから返ってくる情報は、name属性とvalue属性の右辺のみであるから、サーバがこれらの値を受け取り、かつユーザに次の画面を返すには、サーバが最初に発信した、元のページにこれらのユーザから返ってきた情報を埋め込んで、新しいページを再構築して再度サーバはHtmlを発信することになる。このようにあるHtmlのタグにユーザから返ってくるnameとvalueとの新情報を組み込んで、新しいHtmlを構築するプログラムがreconst_html.plである。

このreconst_html.plの基本的な考えは、リストを検索して、ある特定の条

件を満たすタグを見つけ、そのタグの属性情報を書き換えつつ、全体として新しいHtmlを生成するというものである。このsearch/replaceプログラムは、Prologのリスト処理で非常に簡単に実現できる。そのプログラムが本論文に添えられているreconst_html.plである。サーバを構築する、一番の根幹のアルゴリズムがこのソースコードであるが、それはなんと400行足らずのPrologプログラムとして実現されている。

サーバが返す画面は、次のようなものになる。

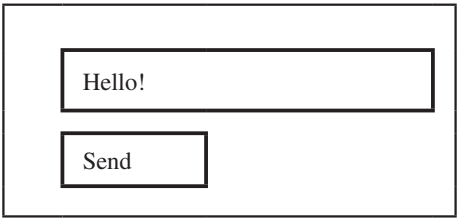

図2

サーバ・プログラムとは、このようなユーザの入力とそれに基づく元のページの更新、再送信というプロセスを繰り返すものということになる。ただ、そのユーザとサーバのやりとりの間に、必要な情報（上の例では、ユーザの入力情報'Hello!'）をハードディスクの中にデータ・ベースとして書き込み、蓄積していくプロセス、それがサーバ・プログラムということになる。つまり、画面上でのやりとり（＝コミュニケーション）以外にも、データを蓄積していくというもう一つの重要な働きが隠されていることになる。

このコミュニケーション・プロセスとサーバが受信したデータを記録するプロセスは、おおむね次のように記すことができるであろう。下記の図の左右の矢印で示されている関係がサーバと端末マシン上のブラウザとのやりとり（コミュニケーション）であり、左半分の処理がサーバの内部処理であり、データを記録するデータベース生成の処理である。

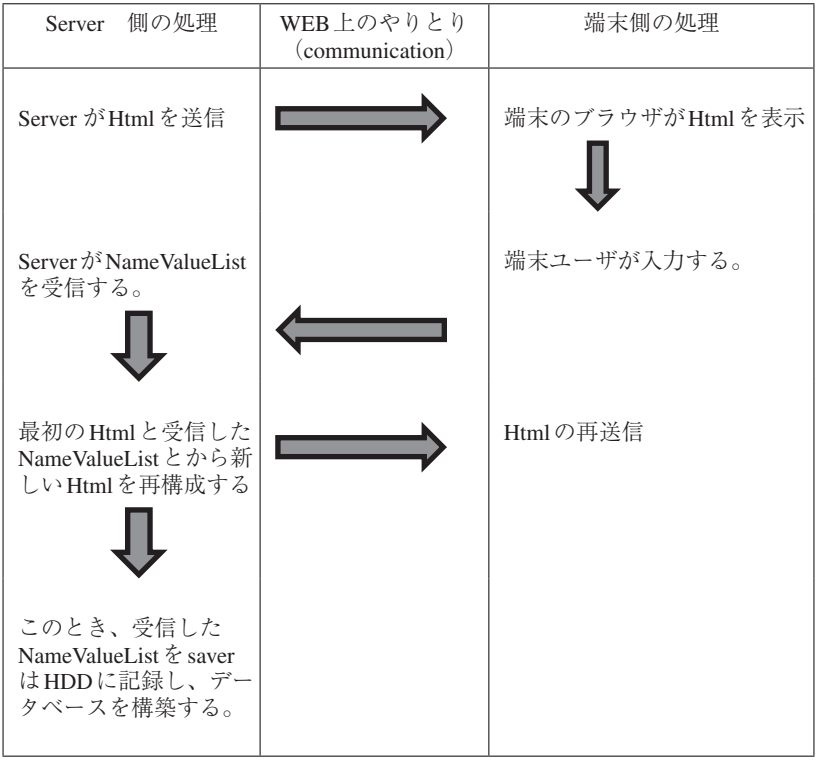

図3

以上のプロセスは、reconst_html.plのプログラムによって処理が実現する。こうして、

① サーバ側と端末側とのコミュニケーション
② サーバ内部でのデータベースの構築

という二つの重要な処理が実現することになった。

7．CGIプログラムは、サーバと端末とのやりとりはただ1回の処理で終わる。

しかし、サーバ･プログラムを構築するには、まだ越えなければならない大きな問題がある。それは、サーバとユーザとのやり取り、すなわちWEB上のコミュニケーション・プロセスは、ただ1回の処理で終了してしまうのである。これが、通常の端末OS上で開発されるアプリケーション･プログラムとの決定的な差であろう。WEB上では、プログラムが自動的には継続しないのである。

この問題に対処するには、端末側のブラウザに常にボタンを用意し、ユーザにそのボタンを押してもらい、これによって処理が継続するという仕組みを用意しなければならない。つまり、CGIプログラムでは、常のユーザの手助け必要となる。ユーザのボタン処理の介在があって初めて、処理が連続的に続けられるのである。

そのために

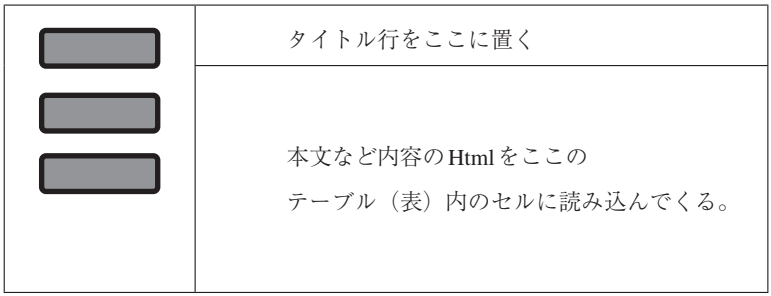

図4

上の図のような左側にいつくかのsubmitボタンを配置し、ユーザがどれか一つのボタンを必ず押すことのできるようなフレームを用意する。ただし、Htmlの最新Ver.ではいわゆるフレーム構造が許されていないので、この枠組みは、むしろ単純なテーブル構造で用意することにする。

こうして、上記のいわば「外枠」としてのテーブル（表）の右下の欄（セル）に実際の内容的なHtmlを組み込んで、合成されたHtmlを作り出すことにする。

そして、左下のコンテンツ枠のHtmlは、サーバに置かれたProlog Htmlのファイルを読み込んで、合成されたHtmlを作ることにする。

すなわち、ブラウザに、データの表示ということに関しては、いわばOSの働きを持たせることにする。ボタンには、サーバのHDDの中のProlog Htmlファイルを探しながら、それらのファイルの名前を上記「外枠」の左下内容セルのなかにファイル名を表示していく。ユーザがそのファイル名をクリックし、左のopen ボタンを押すことによって、当該のProlog Htmlファイルをブラウザ上に表示させるという仕組みを作ることにする。

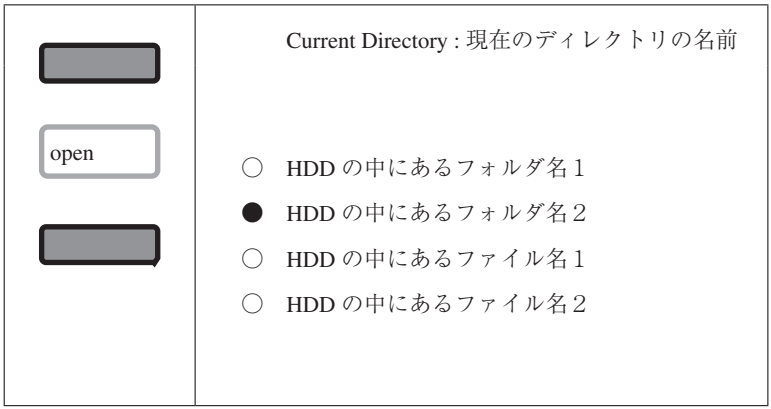

図5

ユーザがopenボタンを押すことによって選択されたフォルダやファイルを開き、さらにそのファイルにsubmitボタンがあれば、そのsubmitボタンにより、ファイルの指定した仕事を実行することにする。

その仕事とは、しばしばテスト問題の自動採点であったり、またVideoショーの再生であたりするだろう。こうしてブラウザをいわばファイル・

マネージャとして構築したサーバがMY Serverであり、これらはすべてPrologというプログラミング言語で書かれている。通常Htmlファイルは Unix/LinuxのOSでも実行形式とは認識されていず、構造化された（＝マークアップmarkupされた）ただのテキスト・ファイルでにすぎないけれども、そのHtmlファイルにCGIのタグ（input, select, textareaのタグ）が含まれていれば、そのCGIタグをある特定の実行プロセスのトリガーとして認識し、コミュニケーションとデータ処理の実行を行う実行ファイルとしての機能を持たせている。

したがって、MY Server上に新たなボタンとその処理をプログラムすることによって、さまざまなコミュニケーション・ベースのアプリケーション・プログラムを開発することが可能になる。したがってMY Serverとは―敢えて大胆な比喩を用いれば―「WEBのブラウザ上に構築する疑似的なOSを実現している」という位置づけをすることが可能と思われる（もちろん本来のOSという意味では到底ないことは言うまでもないが...）。

そしてユーザから返ってくる文字列、あるいは自然言語のテキストをProlog独特の自然言語処理プログラム、あるいは自然言語の構造解析プログラムに載せることを可能にする基盤のソフトウェアと言える。この最後の点にこそ、本論文を敢えて名古屋外国語大学の言語研究の一つとして発表する所以である。

Ⅱ．アプリケーションの開発

１．名古屋外国語大学　日本語教育センター（徳本　浩子）

I.で述べたWEB　OSとしての“MY Server”上で、実務に使えるアプリケーション・プログラムを開発した言語教育の実践者かつ研究者が、名古屋外国語大学　日本語教育センターの徳本浩子である。彼女は、同センターでのOnline Placement Test（OPT）や語彙練習プログラムonline vocabulary practice（OVP）等を作成し、答案の回収と自動採点、結果のExcel

シートへの取りまとめという、一連のオンラインテストの手順を形式化し、かつ実用に耐えるプログラムとして動作させることに成功した。この研究は、徳本が広島大学　大学院総合科学研究科に論文博士申請用として提出した博士号論文（2011年度受理審査合格後、本審査中2011年12月現在）に詳しいので、ここではごく簡単に触れるのにとどめる。

“MY Server”以前の試み：

名古屋外国語大学でのオンラインテストの試みでは、“MY Server”以前では、BlackboardやMoodleといった、汎用のWEBサーバを用いて、オンラインテストを作成する試みが行われていた。これらのオンラインテストの試みでは、1ページあたり50〜60問程度のシートが5〜6枚にわたるオンラインテストでは、教室で学生が一斉にアクセスするとき、メモリに過度の負荷がかかり、それらのオンライン・テストはしばしばフリーズしたり、テスト結果を回収できない事態に見舞われることも多かった。ある意味でこれら汎用のオンライン・サーバでのテスト回収は、教室での一斉使用は、不可能かと思われていた。1ページ当たり数問のミニテストに制限するか、学生に自宅からアクセスさせ、事実上の同時アクセスがない、オンラインテストにせざるを得ない状況が続いていた。

“MY Server”上のPlacement Testの構築（徳本浩子）

これらの経験から、“MY Server”では、各学生が答案提出（submit）し、サーバがデータベースに記録をする際の処理を極小にとどめ、一回一回の学生の答案提出時には、自動採点という複雑な処理を行わないで、全学生の答案の提出が終わった時点で採点とExcelデータベースへの書き込み処理を行わせる処理を行わせるようにした。

この手順の変更は、結果的に大成功であった。“MY Server”は、その上で

開発されたPlacement Testを数十人の同時アクセスを順調に捌いて自動採点の結果をExcelシートの形で返すことができた。

ここでは、この徳本によるインストラクショナル・デザインの特長的な点を2つ指摘しておく。

i. アクセス権の制限

5枚のオンラインテストに厳密な順番をつけ、初めのテストを飛ばして、後ろのテストを先に行うということもできなければ、一度提出したテスト問題は、再びアクセスができないという、テスト問題に厳密な順序付けを導入した。この順序付けは、語学学習用の初心者のためのオンランテスト（とりわけPlacement Test）では必須の機能であるとした。

実は、この機能は、そのための特別なプログラミングを行ってデザインしない限りは、BlackboardやMoodle上の小テストでは実現できない。WEBのコースのトップページに掲載されたWEBページは、順序をつけず自由にいつでも繰り返し受験できるのが、WEB上の情報提供のそもそものコンセプトであるからである。

“MY Server”上のアプリケーションの開発は、Prologプログラミングをしなくても可能な部分が少なからずある。すなわち、i. のような順序付けの制限を持つオンラインテストでも、Htmlの基本的な知識とMY Serverが用意しているOS機能であるアクセス･パミッション（ユーザへのアクセス権限の付与）だけで、アプリケーションが開発できるという性質を持っている。そのため基本的にはHtmlの知識がありさえすれば、MY Server上のアプリケーションが開発可能である。

ii. 内容的な分類

5枚のオンラインテストの各ページが日本語能力のどのような点を測ろうとしているかによって相互に分類されているが、この内容的な分類は、各ページごとにも分類のための仕掛けが用意されている。すなわち、徳本

は、一つのページにおいてもそれぞれの問題ごとに問題の類別を示す問題属性（Attribute）を記入していった。これによって、一つ一つの問題が学生のどの日本語能力を測ろうとしているのかが明示されることになった。

これによって、オンライン・プレイスメント・テスト（OPT）が学生のスコア（総点）のみによるクラス分けではなく、日本語理解の学力の内容的側面からもクラス分けが可能になっていった。そのため、7，8年の実践の中から、徳本のOPTは、効率的なクラス分け試験として機能することが判明した。

徳本の開発したMY Server上のオンラインテストのブラウザ画面上のアピアランス（＝見かけ）をアプリケーション開発者の許可を得て、以下に掲載しておく。

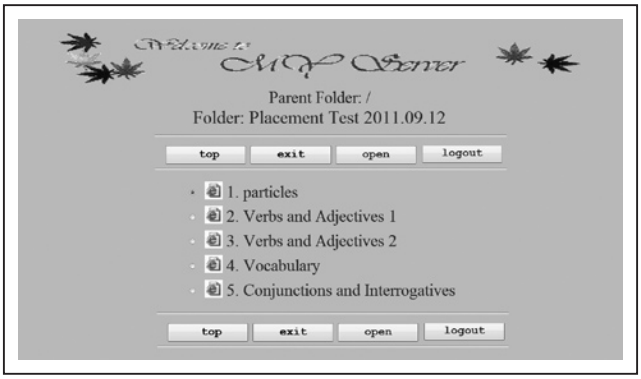

図6

これら5枚のテストは相互に完全に順序付けられており、学生のアクセス後一度答案を提出した後は、もう2度とアクセスできないことになっている。現在の問題フォルダに見られるどれか一つの問題ファイルを選択してopenボタンを押せば、問題文が現れる（ただし、この場合1.から5.の順序で開いていくことになる）。

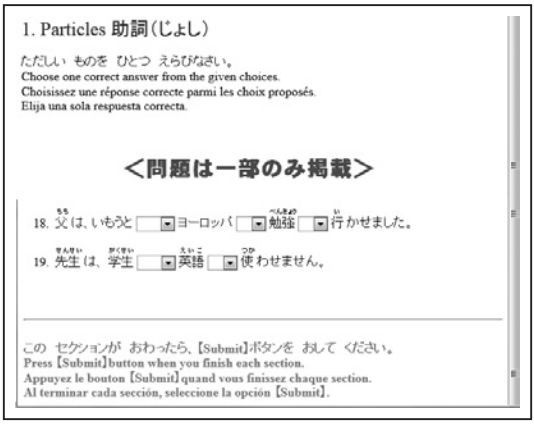

図7

また学生の答案は、内部的には、自動採点のうえ提出答案がそのまま記録されている。

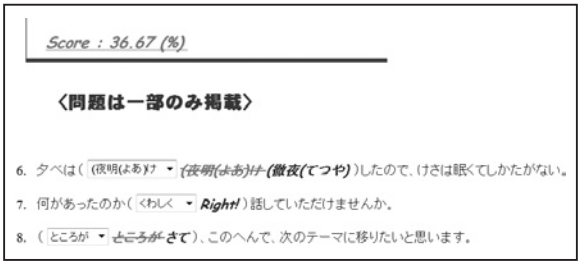

図8

クラス全体の成績は、Excelシートとなってサーバに記録されている。(以下の図表における学生の名前は、仮名)。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	学生番号	姓	名	設問	正答	学生回答	判定	設問有効点	得点	出題課	Category1	Category2	Category3	Category4
2	5077127	Smith	John	Particle_01_01	が	で	X	0	0	29	動詞	自他動詞	～でいるの主格	発見
3	5077127	Smith	John	Particle_02_01	が	が	O	1	1	22	動詞	e	名詞修飾節中の主語	
4	5077127	Smith	John	Particle_02_02	は	は	O	1	1	4	動詞	e	主題	
5	5077127	Smith	John	Particle_03_01	の	の	O	1	1	34	動詞	e	名詞接続	あとで
6	5077127	Smith	John	Particle_04_01	を	を	O	1	1	24	動詞	e	人的目的語	通る
7	5077127	Smith	John	Particle_05_01	を	が	X	0	0	29	動詞	自他動詞	直接目的語	
8	5077127	Smith	John	Particle_05_01	は	を	X	0	0	29	動詞	自他動詞	主題	～ているの主格
9	5077127	Smith	John	Particle_07_01	に	に	O	1	1	37	動詞	受身	動作主	
10	5077127	Smith	John	Particle_07_02	を	を	O	1	1	37	動詞	受身	直接目的	迷惑受身
11	5077127	Smith	John	Particle_08_01	が	を	X	0	0	22	動詞	e	名詞修飾節中の主語	
12	5077127	Smith	John	Particle_09_01	に	に	O	1	1	37	動詞	受身	動作主	
13	5077127	Smith	John	Particle_10_01	まで	まで	O	1	1	4	動詞	e	時間	まで
14	5077127	Smith	John	Particle_10_02	までに	で	X	0	0	17	動詞	e	時間	までに
15	5077127	Smith	John	Particle_11_01	が	が	O	1	1	19	動詞	e	名詞修飾	ことがある
16	5077127	Smith	John	Particle_11_02	と	と	O	1	1	33	動詞	e	引用	と
17	5077127	Smith	John	Particle_12_01	に	に	O	1	1	11	動詞	e	経度	
18	5077127	Smith	John	Particle_12_02	に	に	O	1	1	13	動詞	ます形	移動目的	移動動詞と共起
19	5077127	Smith	John	Particle_13_01	を	の	X	0	0	23	動詞	e	ルート	を
20	5077127	Smith	John	Particle_13_02	に	を	X	0	0	23	動詞	e	方向	に
21	5077127	Smith	John	Particle_14_01	が	が	O	1	1	30	動詞	自他動詞	～てあるの主格	
22	5077127	Smith	John	Particle_15_01	に	に	O	1	1	37	動詞	受身	動作主	
23	5077127	Smith	John	Particle_15_02	を	を	O	1	1	37	動詞	受身	直接目的	迷惑受身
24	5077127	Smith	John	Particle_16_01	の	の	O	1	1	42	動詞	e	名詞接続	のため
25	5077127	Smith	John	Particle_16_02	が	が	O	1	1	13	動詞	可能動詞	主語	ほしい

図9

2．名古屋学芸大学　英語教育（安藤　直）

名古屋学芸大学の安藤　直は、その英語教育の実践研究として、自分自身の語るVideo映像にアンケート用紙を用意して、WEB上で学生とのインタラクティヴなやりとりを実現する教育教材アプリケーションを開発した。また、学生にiPadやiPhoneなどの端末を持たせ、最新の端末を利用したオンラインテスト（任意回数のアクセスを許す）英語学習教材を開発し、

図10

外国語教育メディア学会（LET）などの学会発表（2011年度）を行った。

３．その他　自然言語処理への応用

WEB 上の generative grammar（生成文法）の試み
（名古屋外国語大学　総合教養　（松村　保寿））

“WEB OS”としてのMY Serverの開発者としての松村は、そのシステムそのものがまたPrologというプログラミング言語で書かれているため、自然言語処理が比較的容易である。とりわけ松村（名古屋外国語大学　総合教養）は、N.Chomsky以降に展開されたG.Gazdarらの一般化句構造文法のアプリケーション化を試みている。ただし、この試みは上記の徳本や安藤らの試みとは違って、自然言語処理としての実用的な有用性にはいまだほど遠い現状である。極小の文法生成プロセスをWEB上で視覚的に見せるという教育上の効果はあるとしても、“MY Server”が目標としている自然言語処理への応用はいまだ遠い目標にしか過ぎない。目下のところ“MY Server”が、実用的な有用性を持つ試みとしては、やはり徳本、安藤らにみられる選択肢によるユーザ応答か、テキストの書き込みと正答を限定的に抑えた自動採点プログラムによる教育教材アプリケーションの開発ということになろう。

この結論自体は、確かに“MY Server”の開発以前で既に自明の結論ではあっただろうが、それでも自然言語処理が遠大な目標に設定された、言語学者自身による言語の教育実践と言語研究のための“WEB OS”の開発が意義なしとはしない。

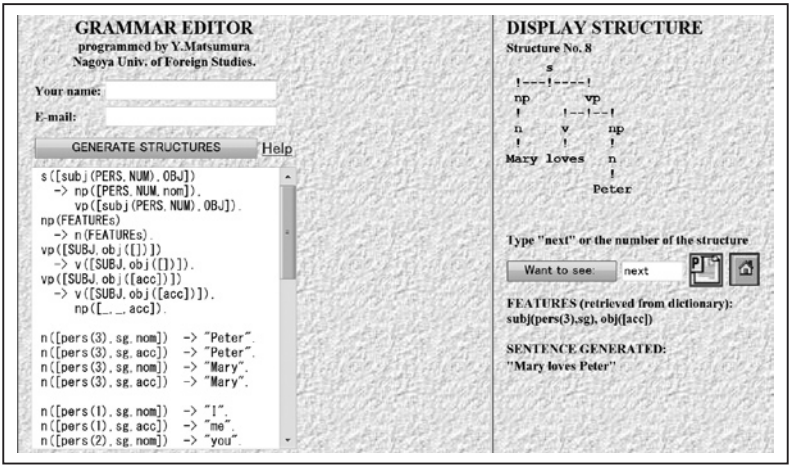

図11

【注】：

基本的には本論文とそれに添えるPrologソースコードの開発にあたって参照した文献はありません。参照にしたものは、唯一オープンソースGNU-Prologコンパイラgprologのドキュメントです。本論文の著者による、添付のソースコード（プログラム）は、すべて著者自身のプログラミングにより産み出されたものです。そしてPrologを拡張して、WEB上のプログラミングを可能にするシステムにするというアイデアそのものもおそらく、世界的にみてもこれまで誰も試みることがなかったと思われます。二度のProlog開発者の世界大会（ポルトガル, ファーロ2006とロシア、サンクトペテルブルク2008）でも新しい試みとして認められ、かつフランスのPrologコンパイラーの開発元（http://www.gprolog.org）でも本研究の成果が取り上げられています。

注

[1] 1972年ごろにフランスのアラン・カルメラウアーとフィリップ・ルーセルによって考案された。

[2] 通商産業省（現経済産業省）が1982年に立ち上げた国家プロジェクトの開発目標である。570億円を費やし、1992年に終結した。

[3] Pure Prologにおける「論理」の原理は、制限された一階述語論理のレベル（Horn節）であり、Montagueの時間と空間に相対化された“Indexical Semantics”は、おそらくPrologプログラミングを行う中で実現し得るに過ぎない。

参考文献

Diaz, Daniel (2011):

(Daniel.Diaz@univ-paris1.fr)

gprologコンパイラーのドキュメント ver.1.4

http://www.gprolog.org

Gazdar, Gerald et al. (1985):

Generalized phrase structure grammar, Blackwell, 1985.

Chomsky, Noam (1962):

Syntactic structures, 'S-Gravenhage : Mouton, 1962.

Chomsky, Noam (1965):

Aspects of the theory of syntax, M.I.T. Press, Mass., 1965.

Montague, Richard (1974):

Formal Philosophy – Selected Papers of Richard Montague –

(ed.) Richmond H. Thomason

New Haven and London, Yale University Press, 1974.

徳本浩子（未公開）：

教育実践の記録としてのサーバ･プログラム構築に関する研究 ―ブレンド型授業の実効性に関する実証的研究を通して― 広島大学　大学院総合科学研究科論文博士　博士号論文（2011年度本審査中 2011年12月現在）

松村保寿（2001）：

自然言語生成/認識プログラムのWEB公開

名古屋外国語大学 紀要 第21号 名古屋外国語大学外国語学部 pp.23-58