DandD の開くニューパラダイム

柴田里程，滝良太，横内大介，林剛正，宮澤美紀 * 小澤和弘 †

平成 12 年 9 月 13 日

1 はじめに

DandD プロジェクトはデータサイエンスの基盤としての「データとその記述の一体化 (Data and Description)」のルールを確立し，その検証と普及に必要なソフトウエアを開発し実用に供する目的で 1998 年度より開始された科学研究費プロジェクトである． 本論文はこのプロジェクトの中間報告として日本統計学会の諸兄姉にその内容と現状を理解していただくためでなく，統計科学が飛躍を遂げる一つの方向を示すものであることを願って執筆された．

2 DandD ルールの実装

DandD ルールでは，データ本体は単純な数値の並びであるデータベクトルの集合としてまとめられており，それらを構造化したり属性を加えたりすることによって複雑な関係を持つデータを作り出している．これは，データ本体である実体 (entitiy)，それらの間の関係 (relation)，その属性 (attributes) を分離することによって柔軟性と汎用性を確保するという，情報科学の最近の流れにそった構成をとっているためである．

このルールの実装手段としてはさまざまなものがありうるが，本プロジェクトでは XML というマークアップ言語を用いて実装することにした．一般にこの言語に従って作成された文書のことを valid XML document という．XML には別に DTD(Document Type Definition) という文書構造を定義することができ，それにも即しているかどうかも問題になる．このような DTD に適合した文書を well-formed XML document と呼ぶ．DandD ルールの実装とはこの DTD を DTD ファイルという形で定義することにほかならない．今後，この DTD を介して定義された DandD ルールに適合した well-formed XML document のことを DandD インスタンスと呼ぶ．インスタンスという用語は， オブジェクト指向言語であるクラスに属するオブジェクトのことをそのクラスのインスタンスと呼ぶことに由来している．

*慶應義塾大学理工学部
†岐阜大学工学部

3 DandD ブラウザ

DandD インスタンスはそれ自身可読なテキストファイルであるが，人間にとって理解しやすい形式ではない．特に DandD では，実体，関係，属性の分離を計っているので，何らかのブラウザ (browser) を介して人間にわかりやすい表示をする必要がある．また，その利用目的に合わせて表示形式を自由に変えることができる点も XML によって記述した効果である．

一般的な XML 文書用のブラウザとしてはすでに InternetExplorer5.0 や NetscapeNavigator6.0 Preview Release 版など Web ブラウザとして広く用いられているものがいくつかあり，それを利用することが考えられる．しかしこれらが内蔵している構文解析プログラムつまりパーサは非検証 XML パーサとよばれるパーサであり， XML の文法に従っているかどうかだけをチェックする．DTD に従っているかどうかはチェックしないため，それをチェックするには別に検証プログラムを用意する必要がある．このような目的のために利用できるプログラムとして，たとえば STILO といったプログラムも容易に入手できるが，実は DandD インスタンスの場合は，構文チェックだけでは十分でない． たとえば， XML では文字と数字の区別をしていないため，数字の間に意味のない文字列が含まれていても構文上は問題ない． つまり DandD インスタンスが正しく記述されているかをチェックするには，構文 (Syntax) と意味 (Semantics) の両面からチェックする必要がある． ただし，この問題はデータ取得の現場やデータの流れの途中で，直接 DandD インスタンスを書き換えたりすることをさけ，常にプログラムを介して生成，更新するようにすればそのかなりの部分を解決できる．

さらに，一般的な XML ブラウザ はどのような処理をして表示をするかという「書式」を指示しないかぎり XML インスタンスをそのままの形で表示するだけである．XML ではこのような処理の指定をスタイルシートと呼ぶ．スタイルシートを記述する方法としては現在 CSS (Cascading Style Sheets) と XSL (eXtensible Stylesheet Language) の 2 通りがある．

3.1 CSS と XSL

CSS は元来 HTML 用のスタイルシートとして開発されたもので，簡易なスタイルシートである．CSS にはレベル 1(CSS1) とレベル 2(CSS2) の勧告があり，CSS2 が XML にも対応している．CSS2 は CSS1 と上位の互換性を持った仕様になっており，いくつかの例外を除けば CSS2 スタイルシートは CSS1 スタイルシートとして機能する．

一方 XSL は XML 用に開発されたスタイルシートで，CSS と同等の機能のほかに独自の書式設定タグやプロパティを作成して言語を拡張することが可能である．また表示の制御機能も CSS より優れており，項目を表示する順番を指定したり，タグの変換をすることも可能である．

そこで DandD プロジェクトでは，CSS よりも文法は煩雑であるが拡張性や後々の改良を考え XSL を採用した． 以下では DandD インスタンス用に作られた XSL を用いたブラウザを DandD ブラウザと呼ぶ．

3.2 XSL

XSL は XML 文書を別の XML 文書もしくは HTML 文書に変換する言語 XSLT (XSL Transformations) と表示形式を整えるための FO (Formatting Object) から成り立っている. XSLT は Version 1.0 が 1999 年 11 月 16 日 に W3C の勧告になっており，XSL そのものについても 2000 年 3 月 27 日に最新の WorkingDraft が公開されている．この意味では世界標準として確立しつつある言語であるが， W3C は協議機関でありそれ自身一つのソフトウエアとして実装する機能は持っていない．機能定義の方が先行している現状では XSLT の仕様に完全に対応している Web ブラウザはない．また，FO に対応している Web ブラウザもまだない. しかし Windows 版の Microsoft Internet Explorer5.0 は Microsoft XML Parser とも呼ばれ，W3C の XSLT に対応し始めており，W3C の仕様に準拠していないが XSLT に近い機能 MS-XSL も独自に実装している．また，Netscape navigator も近い将来 XSL をサポートすることをアナウンスしている．

本プロジェクトでは， W3C に準拠したブラウザならばどのブラウザーでも問題なく処理できることを目標に DandD ブラウザを開発しているが，現状では，多少 MS-XML に依存している部分も残っており，この点の解消は早急に解決すべき問題である．

3.3 DandD ブラウザの設計方針

DandD ブラウザでは XSLT スタイルシート `DandD.xsl` によって DandD インスタンスを HTML インスタンスに変換し，それをブラウザ本来の機能によって表示させている．そのため，構造化された DandD インスタンスをどのような形式でも表現することができる． したがってユーザの必要に応じて表現形式を変えたり不必要な部分を削除したり，逆に必要なヘルプ機能を埋め込むことも可能である．しかし，一般的にはつぎのような方針で設計することが妥当であると考え試作した．

1. 必要に応じて必要な情報を見せる．たとえば，見出しだけを表示し，クリックしてはじめて内容を表示することなどがこの例である．

2. ホモモルフィズム (homomorphism)
   同じ内容でも，その状況に応じて表示形式を変化させる．たとえば， `Data` の各項目では構造化された目で見やすい形でさまざまな属性もわかりやすい形で表示するが，`DataBody` の各項目では数値を直接高速に表示する．

ただし，プロジェクトを進行していくうち以上のことを実現するには XSL だけでは不十分で JavaScript，DOM，Java など別の言語の手を借りなければならないことが判明し，一部これらの言語も使用している．実際 XSLT では単純な構造の HTML 文書にしか変換が出来ない．たとえば，必要な情報を見るために新しいウィンドウを開く，各データベクトルの数値，属性を取り出す，画像を動かすには別の言語を利用する必要がある．このような操作には基本的に JavaScript を用いることにした．また構造化文書に対する API(Application Programming Interface) である DOM (Document Object Model) が XML など特定の言語に依存しない形での文書の内容，構造，スタイルにアクセスできる操作を定義しており XSLT や JavaScript では行えない操作が含まれているので，各ノードの名前や属性，タグ間に書かれているテキストなどを取り出すためには DOM を使用している． さ

らに，ネットワークを介したリモートアクセスには JavaApplet と呼ばれる Java によるプログラムも使用している．

これらについては「テクニカルノート」の節で詳しく説明するが，開発の過程で必要に応じてさまざまな言語を利用したため，基本的には XSLT を用いてはいるものの，多少複雑になったきらいはある．XSLT と Java だけを用いた DandD ブラウザの開発など今後の課題である．

3.4 画像データ

データの背景情報として画像も重要であり DandD ルールも静止画像, 動画像を含められるよう設計されている．その採り入れ方の特徴は，画像データも他の数値データと同等に扱うことであり，このことによって構造の単純化を計るだけでなく，さまざまな形での参照を可能にしている．より具体的には，画像データも DataBody 中の一つの DataVector として定義し，その DataType 属性によって，静止画像であるか動画像であるかを判断している．Image であれば前者，Slides であれば後者である．さらに Format 属性でその画像形式を示すことができる．

動画像は一般的に，MOV 形式や AVI 形式などになっている．これらは一般的に MOVIE ファイルと呼ばれており，.avi や .mov などの拡張子がついている．これらを眺めるだけならばメディアプレーヤーと呼ばれるさまざまなソフトを利用すればよいが，データと画像フレームとの 1 対 1 の対応をつけるのには利用できない． 動画に関する基本技術はまだ発展途上であり流動的でもあるため，本プロジェクトでは動画を 1 フレームごとに分解 (スライス) し，GIF 形式や JPEG 形式のような画像ファイルの集まりとして扱うことにした．現段階では GIF 形式だけを実装している．もちろんこの方法の問題点は各フレームごとにスライスした画像は合計するともとの MOVIE ファイルよりも非常に大きなサイズになることである．将来的には, MPEG 形式などを直接サポートすることを考える必要がある．

スライスした各フレームは「番号 + 拡張子」と名前を付けてディレクトリに保存した．ここで番号は，001 のように付けるとフレーム数を制限することになるため 1,2,3,... とつけることとした． したがって， 静止画像の場合は画像ファイルの存在場所を直接 <DataVector> の URL 属性として与えればよいが，動画像の場合にはスライスしたフレームの存在するディレクトリを与えることになる．拡張子は属性 Format より定めている．いずれの場合も URL で示すので画像データがネットワークのどこにあってもかまわない．

```
<DataVector Id="Slides_of_swim" LongName="水泳データの動画"
            DataType="Slides" Format="gif" URL="swim" Length="145" />
```

このように定義された画像データは DandD インスタンス中でさまざまな形で利用できるが，その形態には大きくわけて次の 2 通りある．

1. <Explanation> での文章中で，画像データの Id を参照して画像を交えた説明とする．この場合は静止画の場合と特に区別することなく対応するデータベクトルの Id を介して引用すればよい．
2. <Data> の関係形式の一部として採り入れることにより他のデータと同期した形で構造化する．スポーツデータなどではこのような形での構造化が後の解析に大いに役立つ．

これらの形態に対応して，ブラウザでの扱いも設計する必要がある．まず，画像データである <DataVector> かその引用がクリックされれば静止画像ならそのまま表示し，動画像ならばスライスした画像をアニメーションで再生して動画のように見せるのが自然であろう．

関係形式の一部として採り入れられている場合には他のデータと同期をとって表示する必要があるが，ここでは関係形式の各記録 (行) に番号をふり，各番号にカーソルを合わせると対応する画像に切り替わるような実装方法をとっている．なお実装の詳細に関しては「テクニカルノート」の節を参照されたい．

3.5 TEX ファイルの処理

DandD インスタンスの Explanation の説明文中では，数式や表を用いた説明が必要になることがしばしばある．これらをどのような形で記述するかについては多様な選択肢があるが，本プロジェクトでは TEX で記述した説明を対象に実装した．この実装方法は TEX 特有ではなく，以下に説明するように XML のパース対象外エンティティの機能を 利用しているだけであるので，原則的にはどのような言語による記述でも同様に実装できる．

TEX のように XML 以外の特定の言語に従って記述さた部分は下に示すように外部エンティティと呼ばれるパース対象外エンティティとして宣言できる．

```
<!ENTITY kaisetu SYSTEM "ファイルの URL" NDATA TEX>
<!NOTATION TEX SYSTEM "TeX を特定するデータの URL" >
```

ここで NDATA や NOTATION は記法 (この場合は TEX) を宣言しており，1 行目の SYSTEM はデータの実体，2 行目は前の記法宣言に対してどのような処理を施すかを処理プログラムの URL で記述する．このように宣言されたパース対象外オブジェクトは文章中の任意の箇所で

```
<Unparsed file= "kaisetu"/>
```

のように引用できる．しかし現在広く用いられているブラウザはまだどれもパース対象外エンティティを処理する機能は実装されていない．そこで， 上記のような宣言とは無関係に処理する方法を実装した．また．TEX の処理系が必ずしもローカルに利用可能であるとも限らない． これらの問題を回避するために，本プロジェクトでは XML スタイルシートでパース対象外エンティティには特別な処理を施し， TEX の処理を処理系のあるリモートマシンに依頼する形式をとることにした．その依頼する方式としては後述のデータベースへのアクセスで用いるのと同じ RMI を用いることにしている．つまり，パース対象外エンティティを表示する必要が生じたら RMI を介して TEX ファイルの存在場所 (URL) をサーバに教え，サーバは TEX ファイルを取得した上で， ブラウザで参照可能なファイル形式，たとえば pdf 形式に変換し，それを臨時ファイルに落す．この臨時ファイルの URL を通知されたブラウザはそれにしたがって変換されたファイルを表示する．これを実現するため, XSL スタイルシートは，たとえば

```
<Unparsed file="kaisetu" URL="http://aloe.stat.math.keio.ac.jp/DandD/gold/memo1.tex"/>
```

というような記述を JavaScript 関数 DisplayTeX の属性 URL を引数とした呼出しへのリンク (リンク名として kaisetu を用いる) で置き換える．関数 DisplayTeX は新しいウィンドウを開き，そこに以下のようなアプレットタグによる記述

```
<applet code = 'TexApplet.class'
 codebase = 'TexApplet.class のコードベース'>
<param name = 'URL' value=' 引数として渡された TeX ファイルの URL'>
</applet>
```

を書き出す関数である．コードベースは Java のアプレットの存在するディレクトリを特定する URL のことである．

次節でこの詳細とコンパイルについて述べる．

3.5.1 TEX のコンパイルについて

まずアプレットはサーバアプリケーションの texMake というメソッドを呼び出し，前節の URL を引き渡す．texMake メソッドは，まず最初に，引数として受け取った URL から TEX ファイルをテンポラリとして用意したディレクトリに臨時ファイルとしてダウンロードする．さらに，この臨時ファイルを後述の tex2pdf という，TEX ファイルを pdf 形式のファイルへ変換するコンパイラに引き渡す．そして最後に，コンパイラによって作成された pdf 形式のファイルを httpd が読めるディレクトリへ移し，ディレクトリの URL をアプレットに通知し，処理を終了する．
ここで問題となるのは pdf ファイルのブラウザ表示だが、一般に pdf ファイルは，AcrobatReader というソフトウェアを用いて表示しなければならない。しかし、このソフトウェアは無償でダウンロードが可能であり、さらにブラウザのプラグイン機能を利用すると，ブラウザが参照しているファイル (拡張子) が pdf ならば自動的に AcrobatReader を起動し，表示することが可能である．この機能を利用することにより、ブラウザは URL をアプレットから受け取り pdf ファイルを表示する。

また、tex2pdf だが、これは上述のように TEX を pdf 形式で出力するコンパイラで、既存のコンパイラをいくつか組み合わせてシェルスクリプトで作成した。tex2pdf は以下のような処理をする。

1 TEX のソースファイルに platex をかける。エラーが出なかった場合は dvi ファイルだけを残し、他のファイルはすべて消す。エラーを出した場合はすべてのファイルを削除し、処理を終了する。

2 dvi ファイルに dvips をかけ ps ファイルに変換する。エラーが出なかった場合は ps ファイルを残し、dvi ファイルを削除する。エラーが出た場合はすべてのファイルを削除し処理を終了する。

3 ps ファイルに ps2pdf をかけて pdf ファイルに変換する。エラーが出なかった場合 ps ファイルを削除し、pdf ファイルだけを残し、処理を終了する。エラーが出た場合はすべてのファイルを削除する。

4 DandD インスタンスからデータベースへのアクセス

DandD インスタンスで記述され構造化されている DataBody の実体が DandD インスタンスの内部にはなく外部のデータベース上に存在する場合にはブラウザは何らかの手段でネットワーク上のデータ本体にアクセスし入手する必要がある．

ここで，アクセス先のデータ形式が問題になる．世の中で用いられているデータ形式は極めて多様で，それらをすべてサポートしようとしたら，DandD ルール自体膨大なものになり頻繁にアップデートしなければなくなる．そこで， 本プロジェクトでは対象とする外部データベースをリレーショナルデータベース (Relational Data Base) とした．リレーショナルデータベースに関しては「テクニカルノート」で簡単に解説してあるので参照されたい．

4.1 Web ブラウザからサーバへのアクセス法

現在，Web ブラウザからサーバへアクセスしてデータを取得するには大きく分けて 2 通りの方式がある． 一つは単純にサーバにリクエストをおくり結果をテキストとして受けとる方式である．

1. CGI (Common Gateway Interface) 方式
   ブラウザからのリクエストごとに Web サーバがそのマシンでプログラムつまりプロセスを実行し，その結果をブラウザへ戻す．

2. サーバーサイドスクリプト方式
   独自のスクリプトやタグを HTML ファイルに記述し，ブラウザのリクエストを Web サーバが実行し，結果をブラウザに返す方式．ただし，モジュールにより Web サーバの機能拡張が必要である．PHP がこの例である．

3. サーブレット方式
   サーブレットと呼ばれる GUI (Graphical User Interface) を持たない Java プログラムであり，ブラウザからのサーブレットの実行のリクエストを受け取った Web サーバはサーブレットエンジンにサーブレットの実行を促し，その結果を受けとりブラウザへ戻す方式．サーブレットを管理するプロセスであるサーブレットエンジンが Web サーバ上に常駐している必要がある．この方式は，CGI 方式と比べてプロセスをそのたびに起動する必要がないため効率が良い．

以上のような方式と対比するような方式が次の方式である．

1. Java アプレット方式
   ブラウザからのリクエストに応じて Web サーバから Java アプレットがダウンロードされ，ブラウザはそれを実行するだけで目的の結果が得られる．

この方式の場合，どのようにサーバとやりとりするかは Java アプレットにすでに記述されているため，ブラウザ側では意識しないで済む，この利点を生かし，本システムでは，最後の方式に加えて別のマシン上で動作する Java プログラムを一体として動作させることができる RMI (Remote Method Invocation) の技術と Java プログラムからリレーショナルデータベースにアクセスするための一つのインタフェース群である JDBC (Java DataBase Connectivity) 技術の 2 つを併用することでサーバへのデータアクセスを実現している．以下簡単に RMI と JDBC について解説しておく．

4.2 RMI

RMI を用いると，一方が持つメソッドを他方から呼び出すことにより，通信を担ういわゆるソケットのコードを記述することなく簡単にクライアント・サーバ環境を実現することができる．この RMI と Java の特長を生かすとクライアント・サーバ間でデータ本体をわざわざ特定の形式での文字列に変換して受け渡す必要がなくなり，プログラムが簡素になり効率も向上する．それだけでなく，RMI がセキュリティー面の厳重なチェックをしてくれることも RMI を用いるときの利点である．

RMI を用いた Java によるクライアント・サーバ環境の中心的な存在が RMI レジストリサービスである．具体的には `rmiregistry` という名前のプログラムでありサーバ側で起動される．RMI レジストリサービスが提供されているマシンつまりサーバマシンで Java（サーバ）アプリケーションを起動するとこのプログラムは自分自身の提供できるサービスを

```
rmi://aloe.stat.math.keio.ac.jp/SqlReadServer
```

のような URL 形式の一意的な名前で RMI に登録する. クライアント側の Java アプリケーションやサーバからダウンロードされたアプレットはこの名前を手がかりにそのサービスとの通信を開始する．そして，サーバアプリケーションに存在するメソッドをあたかも自身の内部に持つメソッドのように参照，実行することができる.

4.3 JDBC

前述のように JDBC は Java プログラムからさまざまなリレーショナルデータベースサーバにアクセスするためのインターフェース群であり，これを介することによってサーバの違いを意識せずにアクセスすることができる．つまり Java と SQL の知識だけでさまざまなリレーショナルデータベースにアクセスすることができる．

JDBC は，Microsoft 社によって提供されている ODBC (Open Database Connectivity) をベースに開発されたため，モジュール構成は ODBC と類似しており JDBC ドライバ・マネージャと JDBC ドライバによって構成されている．ODBC は

1. C と C++からのリレーショナルデータベースへのアクセスを前提にしているため，Java を用いる際には ODBC をインタフェースに利用するのは難しい．
2. ODBC は，リレーショナルデータベースへのアクセスのオプションが複雑で，同じ API でも引数によって実行動作が異なることがある．
3. ODBC ドライバはインストール作業が必要であり，あらかじめアクセスするリレーショナルデータベースの情報を設定する必要がある．

のような問題点がる．一方 JDBC を利用すれば

1. Java をベースに作られているためアプレットやアプリケーションからの JDBC の利用に不具合が少ない．

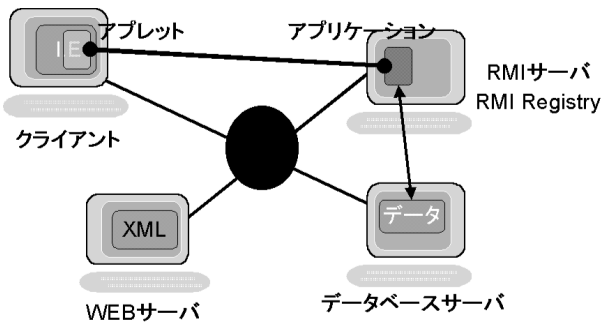

図 1: クライアントとサーバマシンとの関係

2. JDBC ドライバ・マネージャにより，リレーショナルデータベースに依存せずに共通の Java プログラミングが可能である．

3. JDBC ドライバは，必要に応じてネットワークを介してダウンロードが可能であるため，あらかじめインストールすることは必ずしも必要でない．

であるので，本プロジェクトでは JDBC を用いることにした．

4.4 実装

DandD インスタンスをブラウズしている途中で外部データへのアクセスが必要になると，4.4.1 に示す流れでそのデータを取得し，表示することになるが，

1. データベースサーバ

2. RMI サーバ

3. Web サーバ

の 3 種類のサーバがネットワーク上で稼働していることが前提となる．もちろんどの組合せが同一マシン上にあってもよい.

ブラウザの稼働しているクライアントマシンも含めた 4 台のマシンの間の関係は図 1 のようになる．

4.4.1 データベースアクセスの流れ

ここでは，SQL を理解するデータベースソフトにアクセスしデータを取得するまでの流れを説明する．

1. DandD インスタンスを眺めるためクライアントマシン上でブラウザが起動された後，DataBody にデータ自体は含まれておらず，代わりに表 1 に掲げたようなそれを取得するために必要な

さまざまな属性が与えられていれば XSL によって生成された JavaScript プログラムが新しいブラウザを開き, Java アプレットを RMI サーバから取得し，RMI，JDBC を経由してデータベースサーバからデータを取得する．属性は param タグによってアプレットにパラメータとして引き渡される (7.1.4 を参照)．

例 5.1 外部 SQL データ参照のための DandD インスタンスの例

```
<DandD>
<Title>ポテトデータ</Title>
<BackGround>
・
・
・
<DataVector Id="pesticide" LongName="農薬の種類"
Length="120" Controllable="YES"
Nominal="NO" DataType="Category" Code="pesticide_code" DatabaseType="access"
DatabaseServer="172.16.6.110"
SQLCommand="select pesticide from potato"
UserId="DandD"  DatabaseName="dandd" DatabaseSecurity="SECURED"
AppletCodebase="http://172.16.6.110/">
</DataVector>
・
・
・
```

また，UserID と Password が必要なデータベースに対しては，アプレットの入力画面に入力されたユーザー名とパスワードを引数に用い，下記のようにその他のパラメータとともにサーバアプリケーションのメソッド (sqlRead) に渡す．

```
public Vector sqlRead(String command , String user , String passwd ,
String sql , String sqlserver , String dbname)
```

ここで AppletCodebase はアプレットに直接渡されるものではなくブラウザが受け取り，必要なクラスファイルを自動的にダウンロードしたり，通信を制限する SandBox を設定するのに用いられる．

2. 呼び出されたサーバの sqlRead メソッドは DatabaseType に対応する JDBC をロードする．以下にサーバアプリケーション SqlImpl で Database ドライバの選択をする記述を示す．この部分だけが DatabaseType によって処理が変わる．

```
if (sql.equals("postgresql")){
Class.forName("postgresql.Driver");
```

表 1: アプレットに渡す属性

属性	値	必須	記述例
AppletCodebase	アプレットが存在する Codebase	○	http://aloe.stat.math.keio.ac.jp /DandD/server/download/
DatabaseServer	データベースサーバのアドレス	○	aloe.stat.math.keio.ac.jp
DatabaseType	データベースサーバプログラムの種類	○	postgresql
DatabaseName	データベース名	○	DandD
DatabaseSecurity	アクセスの際のデータベースソフトのセキュリティの種別	○	SECURED or UNSECURED
UserId	データベースソフトにアクセスする際のユーザー名	access の場合×	takemasa
SQLCommand	データベースソフトに送信する SQL 文	○	select year from kinri

```
  server = "jdbc:"+sql+"://"+sqlserver+":5432/"+dbname;
}
if (sql.equals("access")){
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
server = "jdbc:odbc:"+dbname;
}
```

3. サーバアプリケーションはロードされた JDBC を通してデータベースと connection を張る．アプリケーションの引数は，server, user, password で，server には，DatabaseServer と DatabaseName が記憶され，user，passwd にはそれぞれブラウザのアプレット画面から入力された UserId と Password が記憶される．

```
connection = DriverManager. getConnection(server , user , passwd);
```

4. SQL コマンドを送信し結果を受け取る．受け取った結果はインターフェースとして準備されている java.sql.ResultSet に格納される．データの取出しはこの ResultSet インターフェースを通して行われ，本プログラムでは取り出したデータを java.util.Vector クラスに順次格納する．結果が 1 列だけの場合と複数列の場合では，アプレット側での表示の方法が異なるため以下のように格納している．

1 列の場合 （1 つの属性に対するレコードのみを表示する場合）要素をそれぞれ element として Vector に格納していき（java.util.Vector.addElement() メソッド），その Vector を返値とする．この場合はすべてのデータを閲覧したいため（DataVector の表示等），データをすべて格納して返値としている。

複数列の場合 （複数の属性を同時に表示する場合）それぞれの行（レコード）を一つの Vector クラスに格納し，それらの Vector を element として持つ Vector クラスを返値とする．複数列を同時に表示するのは，あらかじめ観測値，説明変量，各因子などの概要を見たい場合が多いと思われるため，すべてのレコードを表示する必要はなく 100 行を限度として返値にしている.

このようにして，レコード数，属性数ともに未知のデータを，すべて同じ型 (java.util.Vector) で返している．

5. サーバ側ではデータベースとの接続を閉じ，クライアント側では返値である Vector クラスを，適当に整形してアプレットの画面に出力する．

5 DandD から解析ソフトウエアへのエキスポート

DandD サポートシステムとしては DandD インスタンスを眺めるブラウザーの他にデータ収集現場での DandD インスタンス作成や，すでに存在するデータの DandD 化を支援するためのインポートシステム，さらに，DandD インスタンスからデータを解析のために解析ソフトウエアへ取り込むエキスポートシステムなどが重要である．

本節では DandD インスタンスから解析ソフトウエアにデータをエキスポートするシステムの設計方針と解析ソフトウエア Splus を対象とした実装について述べる．

5.1 エキスポートシステム

エキスポートシステムの実現形態としては次の 2 つが考えられる.

1. DandD ブラウザーに解析ソフトウエアへのエキスポートの機能を取り込みインタラクティブに必要なデータだけをエキスポートする．解析ソフトウエアとの対話も DandD ブラウザを介して行う．

2. DandD ブラウザとは独立に解析ソフトウエア別にエキスポートシステムを用意する．

1. の形態はユーザインタフェイスとしては理想的であるが，現存する解析ソフトウエアには共通なデータ形式がないので, 属性ももたない単純なデータ行列だけですむような場合でないと複雑になりすぎるきらいがある．実際，DandD の役割はこのような解析ソフトウエアの違いを吸収し解析ソフトウエアの違いを意識させないところにもあるため，限定的なエキスポートシステムは望ましくない．また，ブラウザを介してデータを解析ソフトウエアに取り込む方法は 2 パスになるというデメリットも大きい．

以上のことから，2. の形態で解析ソフトウエア別にエキスポートシステムを構築することにした．このようなエキスポートシステムは以前の D&D プロジェクトで構文解析プログラム生成用の Yacc と 字句解析プログラム生成用の Lex を用いて汎用なシステムを作成した経験があるので，これも 2．の形態にした理由である．設計に当たっては，対象とする解析ソフトウエアの扱えるデータ形式の範囲が問題となる． DandD インスタンスは前述の通り XML で書かれており，基本的にリスト (木) 構造である．その木の各節にさまざまな属性が付加されている構造になっている．このような構造を直接扱える解析ソフトウエアならば，エキスポートシステムの設計は単純になるが，いわゆる表計算ソフトウエアのようにデータの扱いに関して機能水準の低いソフトウエアの場合は，エキスポートシステムの負担は極めて大きくなり， DandD インスタンスで記述されたすべてをエキスポートすることは現実的でなくなるおそれもある．

5.2 Splus へのエキスポートシステム

Splus は木構造を直接扱え，さまざまな属性を付加することができるので Splus へのエキスポートシステムは基本的には DandD インスタンスを構文解析し，それを Splus のリストとして移植するソースファイルを生成すればよい．D&D では Splus のダンプ形式を直接生成したが，リストを生成したあとの処理や外部データベースへのアクセスなどを考え柔軟性を考慮し，DandD では S 式からなるソースファイルを作成する方法にした．

Splus へのエキスポートは大きく分けて次の 2 つの過程をとる．

1. DandD インスタンスを Lex，Yacc により字句，構文解析し，木構造をそのまま Splus へ移植するための S 式からなるソースファイルを作成する．

   例 1.1：DandD インスタンス

```
<Title>
        住まいに対する意識調査データ
</Title>
<BackGround>
        <Source Who="M. Madsen(1976)" Date="1988-11-30">
        コペンハーゲンのある地域で 1960 年から 1968 年に渡り
        賃貸住宅の居住者 1681 人について意識調査を行った結果である．
        管理についての意見がどの程度聞き入れられているか，
        その住宅に満足しているかどうか，居住している住宅のタイプ，
        隣人と接触があるかどうかによって居住者が分類され，その
        該当数が分割表の形にまとめられている．
        「現在の住まいのタイプ」の TowerBlocks は高層住宅，
        Apartments は中低層住宅， AtriumHouses は中庭つきのマンション，
        TerracedHouses は連棟式のいわゆるテラスハウスである．
        </Source>
</BackGround>
```

:

例 1.2：S 式からなるソース

```
DandD_list()
DandD$Title_list()
DandD$Title[[1]]_"住まいに対する意識調査データ"
DandD$BackGround_list()
DandD$BackGround$Source_list()
attr(DandD$BackGround$Source,"Who")_"M. Madsen(1976)"
attr(DandD$BackGround$Source,"Date")_"1988-11-30"
DandD$BackGround$Source[[1]]_"コペンハーゲンのある地域で 1960 年から 1968 年に渡り 賃貸住宅の居住者 1681 人について意識調査を行った結果である． 管理についての意見がどの程度聞き入れられているか， その住宅に満足しているかどうか，居住している住宅のタイプ， 隣人と接触があるかどうかによって居住者が分類され，その 該当数が分割表の形にまとめられている．「現在の住まいのタイプ」の TowerBlocks は高層住宅， Apartments は中低層住宅， AtriumHouses は中庭つきのマンション， TerracedHouses は連棟式のいわゆるテラスハウスである．"
      :
```

2. 1．で作成したソースファイルを Splus へ取り込み，データに関する部分の組織化は S 関数で行う．

5.3 Lex, Yacc による構文解析

ここでは DandD インスタンスだけでなく XML で書かれたインスタンスは Splus へのエキスポートを可能とする汎用性を持たせた．ただし，Entity の展開は属性について内部 Entity のみ行う．

構文解析時の注意点を以下に示す．

1. XML の構文法では同一タグ名が複数存在してもよく，実際, DandD インスタンスでも，たとえば複数のリレーショナルデータが存在する場合 Relational というタグ名は共通である．しかし，これでは Splus に取り入れたとき一意的に名前でアクセスすることはできず，データフレームに変換することもできなくなる．一意的な名前をつける方法としては属性 LongName を用いる方法も考えられるが，DandD ルールではこの属性は強制ではないのでいつも存在するとは限らない．したがって，構文解析時に一意的なタグ名に変換する．

2. XML ではデータを示す型に数値型，文字型，文字列型 ... などの区別がなく，みな文字データとして記述されるため，Lex では DataVector に記述された数値ベクトルも Appendix に記述された文字列ベクトルも 1 つの文字列として字句解析する．それぞれの型に合わせた処理は Splus へ取り込んでから行う．

3. XML ではタグ名，属性名，ENTITY 名，ID 型の属性の値に使うことができる文字列に制限がある．先頭に使える文字は Letter クラスの文字と呼ばれる Base Character (アルファベット，平仮名，カタカナなど)，Ideographic Character (漢字など) と "_", ":" であり，2 番目以降の文字は上記の文字以外に Digit (数字)，"." ，"-" である．しかし，"_", ":", ".", "-" は Splus で特別な意味を持つ．そのため，それらの文字がタグ名や属性名に使われた場合は Splus で問題を生じるため該当する文字をすべて "?" に置き換え，Entity 名，ID 型の属性の値に使われた場合は Splus での処理には問題がないためそのままにする．実際には，DandD DTD の定義ではそれらの文字が使われていないためこれらの文字がタグ名や属性名に使われることはない．

4. ID 型の属性の値が重複して存在する場合は，RefId での参照先が一意に決められないため，エラーメッセージを出力し，処理を中断させる．

5. 属性の内容に内部 Entity 参照がある場合は，該当する内容で置き換える．

5.4 Splus でのデータの組織化

DandD ブラウザを併用することを前提に，Splus ではデータの組織化のみ行う．つまり Splus へエキスポートされた DandD インスタンスの木構造のうち，<Data> ，<DataBody> ，<Appendix> のみを用いてデータの組織化を行う．

以下に Splus での組織化の注意点をデータベクトルごとの処理とデータの組織化の際の処理に分けて示す．

5.4.1 データベクトルごとの処理 (<**DataBody**> についての処理)

1. DandD インスタンスでは，データは <Data> で組織化されており，その構成要素となるデータベクトルは <DataBody> の要素となっている．したがって，ID を手がかりにその参照を解決する．

2. <DataVector> の内容がカテゴリカルデータつまり属性 DataType が"Category" あるいは "OrderedCategory" の場合は，属性 Code を参照して S の factor オブジェクトあるいは ordered factor オブジェクトに変換する．また 属性 DataType が "verbLogical"+ の場合は S の論理オブジェクトに変換する．

3. 属性 DataType が"Category" ，"OrderedCategory" ，"Logical" 以外で属性 Code の記述がある場合は，属性 Code を参照して文字列ベクトルに変換する．この場合，属性 Domain に Domain 名の記述があれば<DataVector> の要素がそれぞれ Domain の何番目の要素に対応しているかの識別 (Id) 番号のベクトルも作成する．そのベクトルをデータの組織化の際に "Domain 名 Id" という列として新しく加える．また，Domain 名として記述された Id 名に該当する<DataVector> に対しては属性 Code を参照して文字列ベクトルに変換したベクトルと，変換する前の数値ベクトルを作成し，データの組織化の際に数値ベクトルを"Domain 名 Id" という列として新しく加える．

例 2.1：属性 Domain をもつインスタンス

```
<Data>
     <Relational LongName="愛車">
          <Value RefId="StationByName"/>
          <Value RefId="Year"/>
          <Value RefId="Month"/>
          <Value RefId="Day"/>
          <Value RefId="Liter"/>
          <Value RefId="Price"/>
          <Value RefId="Odometer"/>
     </Relational>

      <Relational LongName="給油所属性">
          <Value RefId="Station"/>
           <Value RefId="Chain"/>
           <Value RefId="Company"/>
           <Value RefId="Place"/>
      </Relational>
</Data>

<DataBody>
       <DataVector Length="31" Id="Station" LongName="給油所リスト" DataType="Id"
              Code="GasStation" >
        1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
        26 27 28 29 30 31
       </DataVector>

       <DataVector Length="244" Id="StationByName" LongName="給油所" DataType="Id"
              Code="GasStation" Domain="Station" >
        19  19  19  19  19  19  19 19  19  19  19  19  19  19  19  19  19
        19  19  19  19  19  19  19  19  ...
       </DataVector>
             :
</DataBody>
<Appendix>
       <Code  Id="GasStation">
         JIRO 洋光台　アカシドウリ　アヅマ商事
         アロピット和田町　カーオアシス旗ノ台　セリガサワ　セント高田
         デリスパーク川崎インター　ポイントビュー 21 葉山
```

```
        モービルニュー等々力　宮崎　元住吉　港北 NT 第一　松本インター
        新城　裾野須山　西部鎌倉ドライビングセンター　西調布
        川崎インター東　双葉 SA　第三京浜口　町田木曽　都筑
        東急柿の坂　東名足柄 SA　白樺　白金　明津　目黒通り　葉山一色
        葉山木古庭
      </Code>
        :
</Appendix>
```

例 2.2：属性 Domain の処理とデータの組織化

```
> gas
$"愛車":
   Julian  StationId    StationByName Year Month Day Liter Price Odometer
 1  13677         19 ’川崎インター東’ 1997     6  12 41.00  89.0    14251
 2  13693         19 ’川崎インター東’ 1997     6  28 48.30  89.0    14516
 3  13706         19 ’川崎インター東’ 1997     7  11 49.20  90.0    14772
 4  13723         19 ’川崎インター東’ 1997     7  28 49.70  90.0    15421
 5  13732         19 ’川崎インター東’ 1997     8   6 48.80  90.0    15781
 :    :           :          :         :       :   :   :     :        :

$"給油所属性":
    StationId              Station        Chain       Company          Place
 1          1          ’JIRO 洋光台’       ’三菱’  ’神奈川菱油’ ’横浜市磯子区’
 2          2        ’アカシドウリ’ ’昭和シェル’    ’新潟昭石’       ’新潟市’
 3          3          ’アヅマ商事’       ’三菱’  ’アズマ商事’     ’八王子市’
 4          4    ’アロピット和田町’       ’不明’  ’矢野新商事’       ’横浜市’
 5          5  ’カーオアシス旗ノ台’ ’昭和シェル’    ’東英石油’       ’東京都’
 :          :              :            :             :              :
```

4. <DataVector>の内容は数値ベクトルに変換し，<Appendix>の内容は文字列ベクトルに変換する．ただし，<Script>での記述がされている場合は内容に合わせて展開する．

5. データベクトルの参照で実体が外部データベースに存在する場合には S 式から Java プログラムを呼び出して実体を取得する．

   - 外部データベースへのアクセス
     データベースアクセスのプロトコルは DandD インスタンス<DataVector>の属性に記述される．これらの記述に従って，S 式から Java プログラムを呼び出す．また，参照する<DataVector>ごとにプログラムを呼び出さなければならない．

データベースアクセスのプロトコル

属性名	記述内容
DatabaseType	データベースサーバーのタイプ postgresql または access
DatabaseServer	データベースサーバーのネットワークアドレス
DatabaseName	データベース名
SQLCommand	データベクトル取り出しのための SQL 文
DatabaseSecurity	セキュリティ UNSECURED または SECURED
UserId	セキュリティが UNSECURED の時アクセスに用いるユーザ名

- プログラム **SqlApli** の呼出し方法

  - セキュリティ：**SECURED** の場合の呼出し方法

    java SqlApli SECURED サーバーのネットワークアドレス データベースのタイプ
    データベース名 SQL 文

    SECURED の場合は，インスタンスに記述されたプロトコルの他にユーザ名とパスワードが必要になる．SqlApli ではユーザ名とパスワードの入力のためのウィンドウを開き入力を要求する設計にした．また，データベクトルごとの呼出しでその都度ユーザ名とパスワードの入力を要求することを避けるために，SqlApli の最初の呼出し時に入力されたユーザ名とパスワードを S 関数内で取り出し，2 度目以降の呼出しに用いることでユーザ名とパスワードの要求をしないようにしている．

  - セキュリティ：**UNSECURED** の場合の呼出し方法

    java SqlApli UNSECURED ユーザ名 サーバーのネットワークアドレス
    データベースのタイプ データベース名 SQL 文

5.4.2 データの組織化の際の処理 (<Data> についての処理)

1. <TimeSeries> および時間情報を持った <Relational>には下記のような変換をした時間情報の新しい列を加える．

時間情報	付加情報名および変換内容
Year, Month, Day	Julian = julian(Month, Day, Year)
Year, Month	YearMonth = Year + Month/12
Hour, Minute, Second	Time = Hour + Minute/60 + Second/3600
Year, Month, Day, Hour, Minute, Second	Julian = julian(Month, Day, Year) + (Hour + Minute/60 + Second/3600)/24

   また，<TimeSeries>で観測時間が規則的ならば S の多変量時系列オブジェクトに変換することも考えたが，実際のデータでは観測時間が規則的なものが少ないことから S の多変量時系列オブジェクトには変換しないことにした．

2. DandD インスタンス<Data>では配列形式<Array>での記述ができる．しかし，Splus では配列データでの解析がサポートされていないため<Array>での軸情報にもとづいた列を付加した関

係形式にしてデータの組織化を行う .

例 **3**：住まいに対する意識調査データ **(housing.xml)**
例 3.1：配列形式での記述インスタンス

```
<Data>
      <Array LongName="feeling">
      <Axis Id="influence"
            LongName="管理に意見がどの程度取り入れられていると思うか"
            AxisType="OrderedCategory" Length="3">
            Low Medium High
      </Axis>
      <Axis Id="satisfaction" LongName="現在の住まいに対する満足度"
            AxisType="OrderedCategory" Length="3">
            Low Medium High
      </Axis>
      <Axis Id="type" LongName="現在の住まいのタイプ"
            AxisType="Category" Length="4">
            TowerBlocks Apartments AtriumHouses TerracedHouses
      </Axis>
      <Axis Id="contact" LongName="他の居住者との接触程度"
            AxisType="OrderedCategory" Length="2">
            Low High
      </Axis>
      <Value RefId="n.person"/>
      </Array>
</Data>
<DataBody>
      <DataVector Id="n.person" LongName="該当人数"
              Length="72" DataType="Count" Unit="1 person">
          21 34 10 21 22 11 28 36 36 61 43 26 23 35 18 17 40 54
          13 8 6 9 8 7 10 12 9 18 15 7 6 13 5 7 13 11
          14 16 3 19 23 5 37 40 23 78 48 15 46 45 25 43 86 62
          20 10 7 23 22 10 20 24 21 57 31 5 23 21 6 13 13 13
      </DataVector>
</DataBody>
```

例 3.2：関係形式での組織化

```
> housing
```

```
$feeling:
   n.person influence satisfaction           type contact
 1       21       Low          Low    TowerBlocks     Low
 2       34    Medium          Low    TowerBlocks     Low
 3       10      High          Low    TowerBlocks     Low
 4       21       Low       Medium    TowerBlocks     Low
 5       22    Medium       Medium    TowerBlocks     Low
 6       11      High       Medium    TowerBlocks     Low
 7       28       Low         High    TowerBlocks     Low
 8       36    Medium         High    TowerBlocks     Low
 9       36      High         High    TowerBlocks     Low
10       61       Low          Low     Apartments     Low
11       43    Medium          Low     Apartments     Low
12       26      High          Low     Apartments     Low
13       23       Low       Medium     Apartments     Low
14       35    Medium       Medium     Apartments     Low
15       18      High       Medium     Apartments     Low
16       17       Low         High     Apartments     Low
17       40    Medium         High     Apartments     Low
18       54      High         High     Apartments     Low
19       13       Low          Low   AtriumHouses     Low
20        8    Medium          Low   AtriumHouses     Low
21        6      High          Low   AtriumHouses     Low
22        9       Low       Medium   AtriumHouses     Low
23        8    Medium       Medium   AtriumHouses     Low
24        7      High       Medium   AtriumHouses     Low
25       10       Low         High   AtriumHouses     Low
26       12    Medium         High   AtriumHouses     Low
27        9      High         High   AtriumHouses     Low
28       18       Low          Low TerracedHouses     Low
29       15    Medium          Low TerracedHouses     Low
30        7      High          Low TerracedHouses     Low
31        6       Low       Medium TerracedHouses     Low
32       13    Medium       Medium TerracedHouses     Low
33        5      High       Medium TerracedHouses     Low
 :        :         :            :           :          :
```

5.5 Splus への実装

DandD インスタンスから S 式のソースファイルを作成するためのプログラムとして XML.l (Lex プログラム) , XML2S.y (Yacc プログラム) から XML2S を作成し , Splus でのデータ組織化のために S 関数 construct.d を作成した . また , XML2S と construct.d を 1 つの操作で実行させる S 関数として DandD2S を用意した . その他エキスポートに関係する S 関数としてデータフレームの表示のための print.Data, as.matrix.data.frame , 文字列として読み込んだ DataVector の値を Splus の中で文字列ベクトルに分ける strtok を作成した . 関係する S 関数についてはすべてファイル Sfuns にダンプし , それぞれの Splus へ取り込むことが容易にできるようにしてある .

5.5.1 Splus へエキスポートするための手順

1. エキスポートプログラムを自分のディレクトリに取り込む

```
% ftp aloe
> get y.tab.c           # Yacc プログラム
> get lex.yy.c          # Lex プログラム
> get Sfuns             # S 関数
> quit
% nmake                 #XML2S ができる
```

2. DandD インスタンスを自分のディレクトリに取り込む

3. DandD インスタンスを Splus へエキスポート

```
> source("Sfuns")       # Sfuns の取り込みをしていない場合のみ実施
> DandD2S("housing.xml")  # housing.xml を取り込む例
```

4. 組織化されたデータを眺める

```
> housing
$feeling:
   n.person influence satisfaction          type contact
 1       21       Low          Low    TowerBlocks     Low
 2       34    Medium          Low    TowerBlocks     Low
 3       10      High          Low    TowerBlocks     Low
 4       21       Low       Medium    TowerBlocks     Low
 5       22    Medium       Medium    TowerBlocks     Low
 :        :         :            :           :          :
```

5. データの解析

```
> attach(housing)
> feeling
   n.person influence satisfaction           type contact
 1       21       Low          Low    TowerBlocks     Low
 2       34    Medium          Low    TowerBlocks     Low
 3       10      High          Low    TowerBlocks     Low
 4       21       Low       Medium    TowerBlocks     Low
 5       22    Medium       Medium    TowerBlocks     Low
  :       :         :           :            :           :

> feeling.glm_glm(n.person ~., poisson, feeling)
> feeling.glm
Call:
glm(formula = n.person ~ ., family = poisson, data = feeling)

Coefficients:
 (Intercept) influence.L influence.Q satisfaction.L satisfaction.Q     type1
    2.991935  -0.3267263  -0.2280383      0.1171634      0.2621952 0.3254572
      type2      type3   contact
 -0.2793184 -0.1027707 0.2154675

Degrees of Freedom: 72 Total; 63 Residual
Residual Deviance: 296.573

> add1(feeling.glm, ~.^2)
Single term additions

Model:
n.person ~ influence + satisfaction + type + contact
                        Df Sum of Sq      RSS       Cp
                <none>              307.0066 394.7227
influence:satisfaction  4  106.3409 200.6657 327.3668
        influence:type  6   16.4675 290.5391 436.7327
     influence:contact  2   17.5800 289.4265 396.6351
     satisfaction:type  6   60.8887 246.1179 392.3114
  satisfaction:contact  2    5.2775 301.7290 408.9377
          type:contact  3   39.6067 267.3999 384.3548
```

以上が基本的な流れである．エキスポートシステムには解析ソフトウエアの使用目的に応じたさまざまな変形がありうる．したがって同じ解析ソフトウエアを対象としたエキスポートシステムでも唯

一とは限らない．さまざまな人々によって改良されていくことを期待している．

6　今後の課題

Splus へのエキスポートシステムの今後の課題に，DandD インスタンス<Analysis> ,<Model> の項の取り入れがある．S 式で書かれているものは実際に使えるように取り入れることを考えている．その他の課題は Splus 以外の解析ソフトウエアへのエキスポートシステムの検討，また DandD インスタンスが DandD ルール (つまり DTD) に従っているかどうかの文法チェックや参照先のない Id 名が RefId に記述されていないかなどの内容チェックができるシステムの検討と実装がある．チェックシステムは DandD インスタンス作成時の単純な入力ミスなどを事前に防ぎ DandD ブラウザやエキスポートシステムでの単純エラーを防ぐ効果が大い．チェックシステムと合わせて必要なのがインポートシステムの検討と実装である．インポートシステムは多くのデータの DandD 化に重要な役目を持つことになるであろう．

7　テクニカルノート

7.1　DandD インスタンスの処理

7.1.1　XSLT

すでに述べたように XSLT は XML ドキュメントを別の XML ドキュメントもしくは HTML ドキュメントに変換する部分を担う言語で，もともとの XML ソースのツリーから表現結果をもたらすツリーへの変換のことである．XSLT による変換は「パターン」と「テンプレート」の組合わせにより実現され，その変換は「スタイルシート」と呼ばれる．つまり，スタイルシートは複数のテンプレートルールの集まりである．

一つのテンプレートルールは例えば以下のような 2 つの部分から構成される．

```
<xsl:template match="Title">
     <DIV>
     <SPAN class="hand" title="click to be visible or invisible details !">
     <A class="symbol">+</A><SPAN class="first"><xsl:node-name /></SPAN>
     </SPAN>
     <SPAN class="displaynone"><xsl:value-of /></SPAN>
     </DIV>
</xsl:template>
```

<xsl:template match="Title"> はソースツリー中の TITLE ノードにマッチしている「パターン」である．この開始タグと終了タグで囲まれた中身が「テンプレート」となる．

XSLT プロセッサが要素や属性を認識できるようにするため，XSL では XML 同様にネームスペースを使用する．XSLT ネームスペースにおける要素は，接頭辞「xsl:」を使用することで XSL ファイル内で認識される．

1999 年 11 月 16 日のネームスペースの URL は「http://www.w3.org/1999/XSL/Transform」，2000 年 3 月 27 日の最新 WorkingDraft では「http://www.w3.org/1999/XSL/Format」となっている．しかしこれらの URL は W3C の仕様であり Internet Explorer 5.0 ではこれらの URL を記述しても機能せず，「http://www.w3.org/TR/WD-xsl」を使用する．また，ネームスペースの URL を勝手に変更したり記述しなかったりすると XSLT プロセッサは XSLT を認識しなかったり，エラーを返す．

以下に代表的な XSL 要素について紹介する．

1. `xsl:apply-templates`
   それぞれのノードの内容，種類にもとづいて適したテンプレートの適応を XSL プロセッサに指示する

2. `xsl:attribute`
   属性ノードを作成し，出力要素にそれを付け加える

3. `xsl:choose`
   `xsl:when` 要素と `xsl:otherwise` 要素を用いて多重条件判断を行う

4. `xsl:comment`
   出力にコメント文を作る

5. `xsl:copy`
   ソースから現在のノードを出力に写す

6. `xsl:element`
   出力に指定した名前の要素を作成する

7. `xsl:for-each`
   ノードの集まりに繰り返しテンプレートを適応させる

8. `xsl:if`
   単一条件によって，テンプレートを適応させる

9. `xsl:otherwise`
   `xsl:choose` 要素と `xsl:when` 要素を用いて多重条件判断を行う

10. `xsl:pi`
    出力に処理指示を作成する

11. `xsl:stylesheet`
    `xsl:template` 要素や `xsl:script` 要素を含むスタイルシートの文書要素

12. `xsl:template`
    特定のタイプや文脈のノードにないし，望みどおりの出力をするためのテンプレートを定義する

13. xsl:value-of
    選択されたノードの中身をテキストとして書き込む

14. xsl:when
    xsl:choose 要素や xsl:when 要素と共に用いて多重条件判断を行う

また Microsoft 社が独自に拡張したスクリプトに対する XSL 要素としては次のようなものがある．

1. xsl:eval
   テキスト文字列を作るスクリプト記述を評価

2. xsl:script
   スクリプト拡張のためのグローバルな変数や関数を定義する

以下は，動画表示を行う XSL プログラムソースである．

```
var win = window.open("","","width=720,menubar=0,resizable=1");
win.document.write(hhh,fcols,"60%,*'>",fname,
"info'>",fname,"data'>",bh,fend);
win.info.document.write("<html><body>");
win.info.document.write("<img src = '",str("URL"),
"/",str("URL"),"0.gif' width='400' height='280'>");
win.info.document.write("</body></html>");
win.data.document.write("<html><head><script language='JavaScript'>");
win.data.document.write("OnMouseB = new Array();");
win.data.document.write("for(var i = 0; i < ",str("Length")," ;i++){");
win.data.document.write("OnMouseB[i] = new Image();");
win.data.document.write("OnMouseB[i].src = '",str("URL"),
"/",str("URL"),"' + i + '.gif';");
win.data.document.write("}");
win.data.document.write("function OnMoSet(flag ,position){");
win.data.document.write("parent.info.document.images[position].src
=OnMouseB[flag].src;}");
win.data.document.write("</script></head><body>");
win.data.document.write("<table border><tr><td></td><td><pre>"
,dimname.fontcolor(t),"</pre></td></tr>");
for (var i=0; i<I ; i++){
var line = "";
for (var j=1 ; j<=J ; j++){
var c = dd[j+","+i];
    if (!isNaN(parseFloat(c)))
       c= Math.ceil(1000*c)/1000;
    line = line+c.toString()+"\t";
```

```
}
    var k= i+1;
win.data.document.write("<tr><td><a href = '#'
target='data' onMouseOver='OnMoSet( ");
win.data.document.write(i + ",0)' >" + k + "</a></td>");
    win.data.document.write("<td><pre>",line,"</pre></td></tr>");
}
win.data.document.write("</table></body></html>");
```

これは win というウィンドウを開き，2 つに分割して，画像とデータを対応させるプログラムを win ウインドウのソースに直接記述している．どのような処理をしているかを以下に簡単に解説する (win.document.write 部分)．

1. まず，win という名前のウィンドウをフレーム機能用いてを左右 2 つに分割
2. 配列 (OnMouseB) の中に画像ファイルを格納し，一枚目を左ウィンドウに表示
3. OnMoSet() という，リンクにカーソル合わせる画像を切り替える関数を作成
4. 最後にデータ番号にリンクを張ったデータの表を作成し右ウィンドウに記述

7.1.2 DOM

DOM は構造化文書 (HTML や XML) に対する Application Programming Interface (API) であり，文書や内容の操作 (要素や属性を含めたすべての文書の内容は，プログラムによってアクセスし，操作できる)，構造のナビゲーション (文書を構成，分解することや，木や要素から他の要素への移動も可能である) ができる．DOM レベル 1 勧告ではどのような構造化文書へもアクセス，編集できるコアインターフェイスと，HTML アクセスのインターフェイスで構成されている．

現在，標準化されている DOM レベル 2 勧告案はレベル 1 の上に構成されており，XML 及び HTML 文書の構造と内容を作成して処理するコアインターフェイスとオブジェクトモジュール群から構成されている．

現在のところ，Internet Explorer 5.0 ，Netscape Navigator 6.0Preview Release 版はどちらも DOM レベル 1 をサポートしている．以下では DOM の記述はすべてこの DOM レベル 1 を指すこととする。

DOM ではドキュメントを構成するあらゆる要素，属性，テキストをオブジェクトとして定義している．どのオブジェクトも処理するためのプロパティ，メソッドを持ち，またオブジェクトはさらに子オブジェクトを持ちうる．

DOM はドキュメントの各オブジェクトの階層を木構造で定義し，そのような木構造の節をノードと呼ぶ．オブジェクトの階層は HTML や XML などのソースにもとづいて生成される．以下に代表的なオブジェクトモデルの型を示す．

Node ドキュメントツリーの節点，葉

Document ドキュメントツリーの概念上のルート (HTML や XML によるドキュメント全体を表している)

Element ドキュメントの要素

Attribute Element オブジェクト内の属性を表す

Text タグで囲まれている中身

Comment コメント文の中身

(補足説明)
Node は一般的な概念で Document，Element，Attribute，Text，Comment はすべて Node として取り扱うことが出来る．

例

```
<elementExample id="demo">
  <subelement1>
  Hello
  </subelement1>
</elementExample>
```

DOM を用いれば最上位のノードは"elementExample"という Element である．最上位の tagName は "elementExample"となる．"Hello" は"subelement1"の Text となる．
DandD スタイルシートでは例えば以下のように使われている．

- xsl:when 要素の条件などして使う
- getAttribute() などのメソッドを用いて属性を取り出したり，文書整形を行う

DandD スタイルシートから上記の例を示す
例

```
<xsl:when expr="this.nodeType==1">
<xsl:eval>this.getAttribute("Id").split(".").join("_")</xsl:eval>
```

1 行目は条件として「"nodeType"が 1 のもの (1 は ElementNode を示している) のとき」ということを記述しており，2 行目では属性 Id を取り出し”.”を取り出して，”_”に置き換えている．

7.1.3 JavaScript

JavaScript は

- シンプルな文法，変数の型が柔軟
- コンパイルの必要がないインタプリタ方式
- オブジェクト指向をベースにしている

- ブラウザ (クライアント側) で処理されるのでサーバの効率が向上する
- インタラクティブなページの作成が可能

などの特徴を持っている．JavaScript のプログラム (スクリプト) は HTML ドキュメントに埋め込む形で記述され，ドキュメントの外観とコンテンツの制御，ブラウザの制御，ドキュメントのコンテンツの操作など様々な Web ページの操作を可能にする．またグラフィック機能，マルチスレッド (並列処理) 機能などはサポートされていないが，簡単な動画再生環境は JavaScript で記述することが可能である．

実際に DandD スタイルシートでは，ドキュメントの整形，新しいウインドウを開く，画像を動かす等の処理を JavaScript で記述した．スタイルシートでの `DisplayData`，`DisplayDataVector`，`DisplayDataStructure` などの`<SCRIPT>`タグ内の一連の関数はすべて JavaScript によって書かれている．

7.1.4 Java

Java は

1. オブジェクト指向のプログラミング言語でありながらシンプル
2. プラットフォームに依存しない独立性で移植性が高い
3. 標準化されたクラスライブラリ
4. 優れたネットワーク機能

などの特徴を持っている．JavaScript は HTML ドキュメントの中に記述することで実行出来るが，JavaApplet はあらかじめコンパイルしたもの (クラスファイルの作成) を用意する必要がある．また JavaScript と違い変数のチェックが厳しく行われる．JavaScript は特定のブラウザ上でしか利用できないスクリプト言語だが，Java は一般のアプリケーションの製作にも利用できるプログラミング言語であり，汎用性が高い．

DandD ブラウザではネットワーク上に存在するデータベース (SQL) へのアクセスや TEX ファイルの処理に Java を使用している．具体的に言うと，DandD インスタンスにサーバ側との通信を行うための記述ある場合 (例えば `DataVector` に属性 `DatabaseType` や `URL` などの記述がある)，スタイルシートは JavaApplet を変換した HTML インスタンス内に埋め込み，RMI を利用してサーバ側と通信を行っている．以下ようなコードでスタイルシートが Applet を変換先の HTML ドキュメントに埋め込んでいる (この例は SQL データベースにアクセスするための Applet である)．

```
if (str("DatabaseType")!=null){
nw.data.document.write("<applet codebase=",str("URL"),
                    "code=SqlApplet.class width=500 height=500>");
nw.data.document.write("<param name='UserId'
value='",str("UserId"),"'>");
nw.data.document.write("<param name= 'DatabaseType'
```

```
value='',str("DatabaseType"),"'>");
nw.data.document.write("<param name= 'DatabaseServer'
value='',str("DatabaseServer"),"'>");
nw.data.document.write("<param name='DatabaseName'
value='',str("DatabaseName"),"'>");
nw.data.document.write("<param name='SQLCommand'
value='',str("SQLCommand"),"'>");
nw.data.document.write("</applet>");
}
```

先頭の if 文は「DatabaseType がある場合」という条件分岐を示している．この条件が真のとき，nw と名付けられたウィンドウの data というフレームに

```
<applet codebase='DandD インスタンス内のコードベース',code="SQLApplet.class"
 width=500,height=500>
<param name='UserId' value='" インスタンス内の UserId"'>
<param name= 'DatabaseType' value=' インスタンス内の DatabaseType'>
<param name= 'DatabaseServer' value=' インスタンス内の DatabaseServer'>
<param name='DatabaseName' value=' インスタンス内のデータベースの名前'>
<param name='SQLCommand' value=' インスタンス内に記述されている SQL のコマンド'>
</applet>
```

という HTML 文を書き出す．

7.1.5 Java アプリケーションとアプレット

Java はプログラミング言語として 1995 年 5 月に Sun Microsystems 社から発表されたオブジェクト指向言語であり，" Write Once Run Anyware" というコンセプトのもと，Java が動くマシンであれば OS を意識せずにプログラムを書くことができるという利点がある．Java の構文は C や C++ などのすでに普及している言語をモデルに作られているため，それらを経験しているプログラマには非常に修得しやすいものとなっている．また Java はインターネットを基盤としてアプレットという形で発達してきた言語であるために通信・ネットワークに強く，セキュリティの概念がしっかり備わっている．またアプレットは，非常に堅固なセキュリティー機能を有している．アプレットは SandBox (砂場という意味) というできることが限られた枠の中でしか動作することができない (表 2)．この SandBox を通してアプレットはコンピュータとやりとりすることになる．一方，アプリケーションにはこのような制限はない.

アプレットとアプリケーションの違いを記述すること あり

表 2: アプレットの SandBox 制限

	アプレット	アプリケーション
通信	ダウンロードしてきたアドレスとのみ通信可能	制限無し
アクセス	ハードディスクやプリンターにはアクセスできない	制限無し
RMI	通信が制限されているためその範囲内でのみ利用可能	制限無し
プログラムの起動	クライアント側のプログラムの起動はできない	可能

7.2 リモートアクセス

7.2.1 サーバが用意するもの

RMI サーバでは，RMI レジストリサービスである rmiregistry，RMI サーバアプリケーション，アプレットをダウンロードさせるための Web サーバを立ち上げる．アプレットはクライアント上のブラウザが読むことができる場所においておく必要がある．rmiregistry は下記のコマンドにより起動される.

```
c:\> start rmiregistry  （Windows の場）合
% rmiregistry &  （UNIX の場合）
```

また用意した JDBC ドライバに対して CLASSPATH は通っていないため，SQL データを取り出す RMI アプリケーションの実行時に，プログラムが JDBC ドライバを読み込めるようにする必要がある．本システムでは，PostgreSQL 用に jdbc6.5-1.2.jar というファイルをダウンロードして使用した ([4] 参照)．サーバアプリケーションを起動するには，次のように JDBC ドライバを CLASSPATH に指定し，実行する.

```
java -classpath /home/user/jdbc/jdbc6.5-1.2.jar:. -Djava.rmi.server.codebase =
 file:///home/DandD/Sqlapplet/  -Djava.security.policy=policy SqlImpl
```

データベースが Microsoft Access の場合は，データベースサーバと RMI サーバが同一のコンピュータ上になければならない.

7.2.2 RMI のセキュリティについて

Java は java コマンドを用いてプログラムを実行する．引数

```
-Djava. security. policy="policy ファイル名"
```

を指定することでアクセスを試みるコードに対し，システムリソースに対するアクセス権を設定できる．これを用いるとファイルアクセスや，ネットワークアクセスについてなど事細かにアクセス権を設定することができる．また，RMI によりサーバ，クライアント間で通信を行う際にデータを暗号化（RSA 暗号）あるいは圧縮する機能も有しており，安全でないネットワーク上の通信や，画像等の大容量データの圧縮などにも対応できるようになっている.

7.2.3 Linux で RMI プログラミングを行う際の注意点

Linux で RMI のサーバアプリケーションを動かす場合には，Windows や UNIX と同じプログラムの書き方では動かない．

```
"connection refused to host:127.0.0.1"
```

というエラーがでてリモートオブジェクトを利用することができない．サーバアプリケーションが RMI レジストリサービスに自分自身を登録する際に（プログラムのソース中では rebind というメソッドで実行される）普通はローカルマシンの RMI レジストリサービスに登録にいくため，URL 中の"rmi://ホスト名/サービス名"の内，サービス名以外は省略することが可能であるはずなのだが，Linux でサーバを起動する場合には明示的に自分自身のアドレスを与えてやる必要がある．理由の詳細は不明であるがこのような不具合の情報が [1] に寄せられている．下記に obj というサービスオブジェクトを SqlReadServer というサービス名で登録する例を示す．

例 2.1

```
Naming. rebind("SqlReadServer" , obj);
```

では Windows や UNIX では動作するが Linux では動作しない．

```
InetAddress localHost = InetAddress. getLocalHost();
Naming. rebind("//"+localHost. getHostAddress()+"/SqlReadServer" , obj);
```

と記述することにより Linux でも動作するようになる．

上の例では，自分自身の IP アドレスを知るためのメソッドを呼び出してそのアドレスに対して rebind を行うことによりこの問題を解決している．

7.2.4 IE5 で RMI アプレットを動かす際の注意点

Microsoft 社のブラウザ Internet Explorer5.0（以下 IE5）では Java が動く，と Microsoft 側は主張しているが，実際 RMI についてはサポートされておらず，RMI を用いたアプレットは動かない．つまりすべての Java アプレットが動作するわけではない．このことについては Sun 側も“ Java が動くという表示を IE5 からはずす”ように求める訴訟を Microsoft に対し起こし世間にも知れるところとなっている（[2] 参照）．

この問題は Microsoft がクラスライブラリから RMI に関するものを削除した事によって生じたものであり，この問題を解決するには削除された RMI 関連のクラスライブラリを，CLASSPATH として OS 側で PATH を通してやる必要がある．このクラスライブラリは，Microsoft によって ZIP 書

庫 (rmi.zip) として提供されており，この書庫ファイルを任意のディレクトリあるいはフォルダに置き，その位置を直接 CLASSPATH 指定すると，IE5 でも RMI のアプレットが動くようになる ([5] 参照)．NetscapeNavigator については，デフォルトのままで RMI が動くようになっている．

7.3 JDBC

JDBC は，Access，SQL Server，Informix，DB/2,IDS Server，Oracle，PointBase，Sybase，PostgreSQL，MySQL 等のほとんどのリレーショナルデータベースに対応しており，動作に必要な各リレーショナルデータベースの固有ソフトウェアも配布されている．また，JDBC のバージョンは，JDK のバージョンに従っており jdk1.1.x に付属されているものは JDBC1.2，Java2 Platform に付属されているものは JDBC2.0 となっている．

7.3.1 JDBC ドライバ・マネージャ

JDBC ドライバ・マネージャは，Sun Microsystems 社から提供されている Java Developer's Kit(JDK) に含まれ，Java アプリケーションは，この JDBC ドライバ・マネージャが定義している JDBC API を利用してリレーショナルデータベースへアクセスを行う．ここで，JDBC API の中で重要である 4 つのクラスを紹介する．

1. java.sql.DriverManager クラス
   リレーショナルデータベースごとに必要な JDBC ドライバの読み込み，リレーショナルデータベースへの接続，java.sql.Connection 型のオブジェクトを返すメソッドなどを実装している．

2. java.sql.Connection クラス
   リレーショナルデータベースへの接続やトランザクションを管理し，java.sql.Statement 型のオブジェクトを生成するためのインタフェースを定義する．

3. java.sql.Statement クラス
   リレーショナルデータベースでの SQL 文の実行および結果取得 (java.sql.ResultSet) のためのインタフェースを定義する．

4. java.sql.ResultSet クラス
   SQL 文の実行結果の処理のためのインタフェースを定義する．

7.3.2 JDBC ドライバ

JDBC ドライバは，JDBC API を構成するインタフェースとデータベース URL 用ドライバで使用する java.sql.Driver クラスを実装している．この java.sql.Driver はリレーショナルデータベースごとに異なるため，どのリレーショナルデータベースにも利用可能で汎用的な JDBC ドライバではない．そのため，JDBC ドライバは，リレーショナルデータベースごとに必要となり，リレーショナルデータベースへの処理方法により 4 つのタイプに分類されている．ここで，それら 4 タイプのドライバを紹介する．

表 3: リレーショナルデータベースと JDBC ドライバ

リレーショナルデータベース	JDBC ドライバタイプ
Cloudscape	2
Hypersonic SQL	2
IBM DB/2	1,2,3
informix	1,4
InstantDB	2
MS Access	1
MS SQL-server	1
MySQL	1
Oracle	1,2,4
Pointbase	2
PostgreSQL	1,2
Sybase	1,3,4

1. JDBC-ODBC Bridge ドライバ
   Java プログラムの実行した JDBC API を ODBC API へ変換し，ODBC ドライバに渡す役割があり，アプレットからの実行はできない．

2. Native API partly-Java ドライバ
   リレーショナルデータベース固有のドライバソフトを利用し，JDBC API をリレーショナルデータベース固有の API へ変換させる．このタイプもアプレットからの実行ができない．

3. Net Protocol All-Java ドライバ
   クライアントとリレーショナルデータベース間に中継サーバを設置して，そのサーバにドライバを置くことにより，クライアントと中継サーバ間でのリレーショナルデータベースに依存しないアクセスが実行となる．

4. Native Protocol All-Java ドライバ
   タイプ 2 のリレーショナルデータベース固有のドライバを Java で作成し，JDBC ドライバ内部に実装させている．

それぞれのリレーショナルデータベースにおける JDBC ドライバのタイプを表 3 に示す．

本システムでは，リレーショナルデータベースが Access の場合には，タイプ 1 の JDBC-ODBC Bridge ドライバを利用した．また，リレーショナルデータベースに PostgreSQL を利用した場合には，タイプ 2 の Native API partly-Java ドライバを利用した．これらのタイプでは，データベースサーバと Web サーバが異なる場合には，アプレットのアクセス制限により Java アプレットでのアクセスが不可能であるが，RMI を利用することにより，Java アプレットからのリレーショナルデータベースのアクセスを可能とした．

7.3.3 JDBC，リレーショナルデータベース間の接続処理

JDBC を利用したリレーショナルデータベースへのアクセスの流れは以下の通りである．

1. java.lang.Class クラスの forName() メソッドを利用して JDBC を読み込む．
2. Java.sql.DriverManager クラスの getConnection() メソッドを利用してリレーショナルデータベースと接続をする．引数に，URL，ユーザ名，パスワードを指定して getConnection() メソッドを呼び出すと，java.sql.Connection 型のオブジェクトが得られる．
3. java.sql.Connection 型のオブジェクトのインスタンス・メソッドである createStatement() メソッドが得られる．さらに java.sql.Statement() メソッドを実行させると java.sql.Statement 型のオブジェクトのインスタンス・メソッドである excuteQuery() メソッドが得られる．これを SQL コマンドを引数として実行させるとリレーショナルデータベースにて処理が行われる．この結果，verb+java.sql.ResultSet 型のオブジェクトが得られる．
4. java.sql.ResultSet 型のオブジェクトから next() や getString() などのメソッドを利用することにより，実行結果を引き出す．
5. これまでの処理で使用したオブジェクトの開放を close() メソッドにより実行する．

7.4 リレーショナルデータベース

現在，データベースはさまざまなデータの管理に利用され，情報の管理には不可欠なシステムである．データベースは，広義には，多量な情報の集合体と考えられ，狭義では，情報産業の中で情報の管理システム，DBMS(DataBase Management System) と呼ばれるデータベースの管理システムと考えられる．現在使用されている主なデータベースは，主にリレーショナルデータベース (RDB，Relational DataBase) であり，以前の網型データベースや階層型データベースよりもシステムの設計，修正が容易である．リレーショナルデータベースは，1970 年に IBM 社の E.F. Codd が発表した論文 [18] に基礎理論が提唱されている．リレーショナルデータベースの最大の利点は，データ構造を表 (テーブル) で表すことにより技術者にとってシステムを容易に理解できることにある．この表は「行 (レコード)」の集合により構成され，行は「列 (属性)」により構成される．列には個々のデータの名前と型を割り当て，数値データの場合には int 型や float 型，文字データには char 型等の型を決める．下表は 6 回のブロック実験の結果をテーブルに入力したものであり，肥料，水量，品種の要因による収穫量を表している．例えば，肥料 1，水量 2 ，品種 4 の収穫量は 10 となる．これらのテーブル名および属性の間に関係 (relation) が定義されたテーブルの集合がリレーショナルデータベースである．

これらリレーショナルデータベースにおける表の処理は，SQL(Structured Query Language) と呼ばれる標準的な操作言語で操作，管理を行う．SQL は，リレーショナルデータベースと独立に存在するため，SQL 文を利用してリレーショナルデータベースに依存せずにデータ処理をすることができる．また，簡単な英単語の並びで記述できるために理解が容易であり，リレーショナルデータベースには SQL が必須な言語となっている．SQL は，1970 年の E.F. Codd の論文発表以後，リレーショナルデー

表 4: テーブルの例

jikkenn

肥料番号	水量	品種番号	収穫量
1	1	1	11
2	1	2	13
3	1	3	14
1	2	4	10
2	2	5	9
3	2	6	12

hirryou

肥料番号	肥料
1	A 社の甲製品
2	B 社の乙製品
3	C 者の丙製品

タベースに関する技術理論や設計，データベース処理言語などの研究が盛んに行われ，多種に及ぶ処理言語の研究の中で 1974 年に IBM 社のサンノゼ研究所により SEQUEL として SQL の基礎が開発された．その後，1977 年に IBM 社のリレーショナルデータベースである SYSTEM-R に SQL として組み込まれた．そして，さらなる標準化の作業が行われ，1986 年 10 月に ANSI (American National Standard Institute：アメリカ国家規格協会) に規格化されリレーショナルデータベース言語 SQL として制定された．さらに 1987 年 10 月には，ISO (International Organization for Standardization：国際標準化機構) が ISO 9075 として制定し，これにより同 11 月に JIS (Japan Industrial Standards：日本工業規格) も X3005 として制定した．現在では，SQL99 が最新の規格として制定されている．これらの団体により制定された SQL は，標準 SQL (一般に SQL と呼ばれるのは標準 SQL である) または SQL 規格と呼ばれ，様々なデータベースアプリケーションに搭載されている．しかし，各データベースアプリケーションには，標準 SQL (今後，単に SQL と呼ぶ) 以外に各ソフト独自に開発された言語も備えているのが普通である．

現在，SQL 準拠のリレーショナルデータベースは数多く，例えば商用では，Oracle，Sybase，informix，SQL-Server，Access，PointBase，InstantDB，IBM DB/2 Cloudscape 等のリレーショナルデータベースがあり，一方，無料で配布されている PostgreSQL，MySQL がある．

SQL によるデータベースの操作には，主に以下の 4 コマンドを利用することになる．

1. SELECT ：データベースのテーブルから条件に合う行を選択する．「問い合わせ」や「クエリー」とも呼ばれ，SQL の最も基本のコマンドである．
2. INSERT ：テーブルに新しい行を追加するためのコマンド．
3. DELETE ：テーブルから行を削除するためのコマンド．
4. UPDATE ：既存のテーブル上のデータを変更するためのコマンド．

先ほどの表 4 を例にして，SELECT コマンドの利用例を示すと，

```
SELECT hiryou.肥料,jikken.収穫量 from jikken,hiryou where jikken.水量=1 and hiryou.肥料番号=jikken.肥料番号;
```

表 5: SQL の実行例

肥料	収穫量
A 社の甲製品	11
B 社の乙製品	13
C 社の丙製品	14

の検索結果は表 5 のようになる．

7.4.1 データベースサーバの起動

本システムでは，リレーショナルデータベースに Microsoft Access2000 と PostgreSQL6.5.3 を利用した．Access は，Microsoft の Windows プラットフォーム上で動作し，個人でデータベースを利用する場合に最も利用されるリレーショナルデータベースである．PostgreSQL は，サポートするプラットフォームが Linux や FreeBSD，Solaris など多数あり，商用・非商用を問わずに無償で利用できる．PostgreSQL は，1986 年カリフォルニア大学バークレー校 (UCB) の Stonebraker 教授により Postgres の呼び名で研究が始められた．当初，データベース処理言語に SQL とは異なる PostQUEL を利用していた．その後，研究の中心人物は次々と変わり，1997 年 1 月に呼び名もメーリングリスト上での投票の結果，現在の PostgreSQL に改名された．PostgreSQL は，今でも開発が続けられており，2000 年 5 月にはバージョン 7.0 がリリースされ，フリーのデータベースとして高く評価されている．

本研究では，Windows，Linux や FreeBSD，Solaris 等の利用率の高いプラットフォーム上での動作を想定しているため，これらのプラットフォーム上でのユーザのリレーショナルデータベース構築と入手，利用の容易性を考慮し，Microsoft Access と PostgreSQL のリレーショナルデータベースを利用した．

1. **PostgreSQL**

   PostgreSQL は，フリーのリレーショナルデータベースであり，ネットワーク上からダウンロードができる (http:// www.postgres.org/ sites.html を参照)．PostgreSQL には，データベースにアクセスするためのユーザの概念が存在し，WWW サーバが PostgreSQL にアクセスするためには，アクセスするユーザが PostgreSQL に登録されていなければならない．さらにネットワークおよびユーザ認証におけるセキュリティポリシーを `pg\_hba.conf` ファイルにより設定ができる．postgres サーバでは，postmaster を daemon として起動することにより動作を開始する．postmaster は，ネットワーク通信の管理および，共有メモリやセマフォなどのシステム資源の管理を行う．

   JDBC ドライバは，ネットワークから入手できるが，PostgreSQL のバージョンに対応したものでなければならない．または，PostgreSQL のソースをコンパイルすることにより入手することができる．

2. **Microsoft Access**

   Accessのデータベースを利用するためには，ODBCとJDBCを併用する方法が取られる．ODBCドライバはAccessをインストールすれば同時にインストールされるが，データソース(DSN)はコントロールパネルのODBCデータソースアドミニストレータから設定できる．

   Accessに限らず，一般にJDBC-ODBC Bridgeドライバを利用する場合は，各リレーショナルデータベースのODBCドライバが存在すれば，jdkに含まれるJDBC-ODBCドライバを利用してリレーショナルデータベースへアクセスすることが可能である．

参考文献

[1] 宮坂雅輝 他「*XML+XSL* による *Web* サイトの構築と活用」SOFTBANK

[2] XML/SGML サロン 「標準 *XML* 完全解説」技術評論社

[3] 岡蔵龍一 他「*HTML&JavaScript* 辞典」 秀和システム

[4] Kazuko Ito 「*Design and Inplementation of Browser for Data and Description*」 慶應義塾大学理工学部修士論文

[5] W3C 「*XSL Transformation Version 1.0*」 URL:http://www.w3c.org/TR/1999/REC-xslt-19991116

[6] W3C「*Extensible Stylesheet Language Version1.0*」URL:http://www.w3c.org/TR/2000/WD-xsl-20000327

[7] W3C 「*DOM Level1*」URL:http://www.w3c.org/TR/REC-DOM-Level-1

[8] W3C 「*DOM Level2*」URL:http://www.w3c.org/TR/REC-DOM-Level-2

[9] RMI on Linux
http://forum.java.sun.com/forum?14@@.787c168d

[10] Sun による訴訟
http://www.sun.com/smi/Press/sunflash/9710/sunflash.971007.10.html

[11] rmi.zip
ftp://ftp.microsoft.com/developr/MSDN/UnSup-ed/

[12] PostgreSQL JDBC Driver Homepage
http://www.retep.org.uk/postgres/

[13] IE で RMI を動かす
http://www.lockwood.org/rmi/

[14] 菊田英明, 実践 JDBC Java データベースプログラミング術, オーム社.

[15] 中村　正弘, JDBC2.0 API リファレンス＆サンプル, 秀和システム.

[16] 糸魚川　茂夫, FreeBSD/Linux で使う PostgreSQL 詳細, オーム社.

[17] 技術評論者編, JAVA PRESS Vol.13, 技術評論社.

[18] E.F. Codd, Relational Model of Data for Large Shared Data Banks, Communication of the ACM vol.13,No.6, 1970.