データベースシステム 108－3
（1996. 5. 23）

グラフィカル問合せ言語DUO用インタプリタの設計・製作

神田儀道　宝珍輝尚　都司達夫
福井大学　工学部　情報工学科

グラフィカル問い合わせ言語DUOの問い合わせの記述を解釈するインタプリタの設計および製作について述べる。特に、DUOの最大の特徴のひとつである括弧を用いた再帰問い合わせを対象とする。

本稿では、実装したインタプリタのユーザインターフェイスについて述べ、問い合わせ記述の変換アルゴリズムについて処理の流れを示す。利用者に使いやすいユーザインターフェイスとするため、グリッドの採用、編集パネルによる編集制限、詳細指定モードの採用を行っている。また、グラフィカルな問い合わせ記述は、DUO表現と呼ぶ中間言語に変換し、DUO表現を問い合わせエンジンで受理される構文に変換する。

Design and Implementation of an Interpreter for the Graphical Query Language DUO

Norimichi KANDA　Teruhisa HOCHIN　Tatsuo TSUJI
Dept. of Information Science, Fac. of Eng., Fukui University

This paper describes the user interface of the interpreter for the graphical query language DUO, and the processing of query expressions. We especially discuss about the recursive query by using a graphical brace, which is one of the major features of DUO syntax.

In order to give a user friendly interface, a grid on the drawing area, editing panels, and the stepwise specification mode are introduced. A graphical query expression is converted into the intermediate language called a DUO expression. A DUO expression is further converted into rules executed in the query processing engine.

1 はじめに

近年、計算機環境は進歩し、手軽にグラフィックスが利用できるようになった。それに従い、今までの分かりにくいテキストのデータベース問い合わせ言語に代わる、グラフィックスを用いたグラフィカル問い合わせ言語が数多く提案されている。著者らも以前よりグラフィカルなデータベース問い合わせ言語として DUO を提案している [1]。この DUO は、初心者にも理解しやすい言語として設計されており、括弧を用いた再帰検索表現が特徴のひとつである。

本論文では、この DUO の問い合わせ記述を解釈するインタプリタの設計及び製作について述べる。ユーザに理解しやすく、かつ、使いやすいインタープリタとするために、グリッドの採用、編集パネルによる編集制限、詳細指定モードの採用を行っている。また、実装にあたり、問い合わせエンジンに演繹データベースシステム Quixote を使用する。グラフィカル問い合わせ記述は、DUO 表現と呼ぶ中間言語に変換し、DUO 表現を問い合わせエンジンで受理される構文に変換する。

以下、2では DUO について概説する。3ではインタプリタの設計仕様について述べる。4では実装したインタプリタに採用したデータ構造と変換アルゴリズムについて述べる。5では実行例を述べる。6ではインタプリタについての考察を行う。

2 DUO について

2.1 概要

DUO の問い合わせの基本は問い合わせグラフである。問い合わせグラフは、要素 (点と枝) と括弧で表現される。例えば、「都市の連絡データから『nagoya』の下り方面に存在する都市を求める」場合、図1のように記述する。

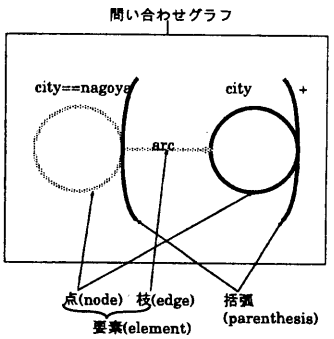

図1 DUO の問い合わせ記述

図1では、nagoya という city から1以上の arc によって連結された city を求めている。図中の"+"は括弧 (()) で囲まれたパスが1以上繰り返されることを示している。また、返答を希望するものは図中の右の点のように太線で、そうでないものは他の要素のようにグレーの線で記述される。

2.2 DUO 表現

ここでは、DUO 表現と呼ぶ、問い合わせグラフを表現するフォーマルな記述について述べる。

2.2.1 要素

問い合わせグラフの基本となるものである。要素には次の2種類が存在する。

点　グラフ中で、丸で表現されたもの。

枝　グラフ中で、矢印で表現されたもの。点どうしの関係を表現する。

グラフ中で要素は、返答を希望するものは図1の右の点のように太線で、そうでないものは他の要素のようにグレーの線で記述される。

検索条件は、その要素のラベルに続けて値を記入することで、指定される。図1の左の点では、「nagoya なる city」を表している。

要素のセマンティックスは6つ組 (et,is,es,rc,V,r) で表現される。ここで、et, is, es, rc, V, r は以下の通りである。

et　要素の種別 (点・枝または逆向きの枝)
is　一意識別子に関する条件
es　要素名に関する条件
rc　検索条件
V　変数の集合
r　返却結果を表す算術式

2.2.2 括弧

括弧は DUO の大きな特徴を表すものである。右の上隅に「+」または「*」を記入することにより、再帰を用いた検索の指定を行う。図1では、「+」の記述により、括弧内のパスを1回以上つないだパスが括弧の評価となる。

また、内部にパスを並列に記述することにより、パスの論理和を指定できる。

括弧のセマンティックスは5つ組 (E,cs,V,VA,prc) で表現される。ここで、E, cs, V, VA, prc は以下の通りである。

E　要素・括弧またはそれらの列の集合
cs　*,+の指定
V　変数の集合
VA　値割り付けを持つ変数の集合
prc　パスに関する検索条件
(count,product,sum,max,min,average)

2.2.3　問い合わせグラフ

問い合わせグラフのセマンティックスは組 (GP、GR) で表現される。

GP　要素・括弧またはそれらの列の集合
GR　問い合わせ結果となる要素または要素列の集合

3　インタプリタの設計

動作環境を以下とする。

- Xwindow 上で動作
  X-window は Unix 上で動作する最もポピュラーなグラフィックス環境であり、OSF/Motif 等の開発環境も整っているため、本システムではこれを使用する。

- DBMS として Quixote システムを利用
  Quixote は ICOT(財団法人新世代コンピュータ技術開発機構) で研究・開発された知識表現言語である [5,6]。今回は、簡単のため、Quixote のシステムの中から micro-Quixote を使用する。

設計指針を以下に示す。

- 使いやすいインターフェイスの設計
  DUO の利用対象が問い合わせ言語についての知識の無い初心者であるため、インターフェイスも使いやすく理解しやすいものであることが必要である。

- 効率の良い変換アルゴリズム
  インタプリタであるため、変換効率の悪さが直接作業効率に反映するので、効率の良い変換アルゴリズムが必要である。また、今後 DBMS を他のシステムに変えたい場合、作業に必要な手間が出来る限り少ない方が良い。

システム構成を図 2 に示す。

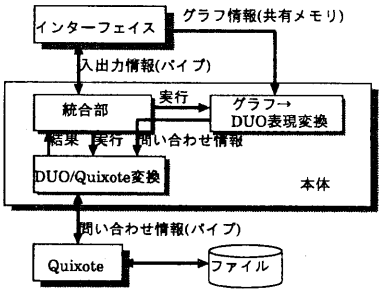

図 2　システム構成

システムは大きく「インターフェイス」「本体」の 2 つに分けられる。本体は更に「グラフ→ DUO 表現変換部」「DUO 表現→ Quixote 変換部」「統合部」の 3 つに分けられる。

本体とインターフェイスは 2 種類の通信方法を使って情報のやりとりを行なっている。一つはパイプで、一般の入出力情報を文字列として送る際に使用する。もう一つは共有メモリで、グラフに関する座標その他の情報を本体に渡す際に使用する。

本体と Quixote との通信もパイプを使い、変換結果を標準入力から読み込ませ、標準出力から出される結果を取り込んでいる。

インターフェイスの外見と各部の名称を図 3、図 4 に示す。

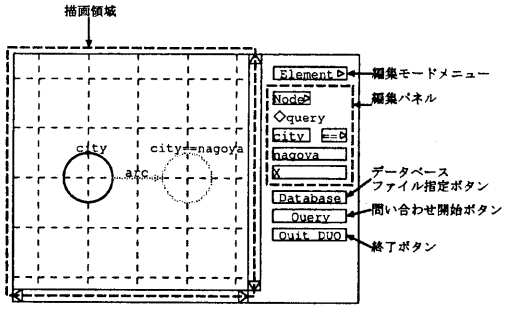

図 3　各部の名称 (外見)

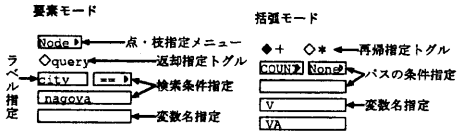

図 4　各部の名称 (編集パネル)

4 インタプリタの実装

4.1 使用したデータ構造

4.1.1 グラフ情報

グラフィカルな問い合わせの記述(グラフ情報)に関するデータはそれぞれの部品に対し3つに分けて定義した。これについて以下に述べる。

1. 各部品毎の数のデータと部品の座標データのあるアドレス

   このデータ型は、繰り返しの処理の際に必要なデータを格納している。NODE_LIST_TYPE、EDGE_LIST_TYPE、PAR_LIST_TYPEが存在するが全て以下の構造を持つ。

```
struct{ int     number;
        void    *start; } …;
```

2. 部品の座標のデータとその部品固有のデータのあるアドレス

   2番目のデータ型は、座標の位置による関係の導出の際に必要なデータを格納する。このデータ型は点・要素・括弧とも異なったデータを持っている。

```
struct npt{
        int             xx, yy;
        ELEMENT_DATA_TYPE *data;
        struct npt      *next; };
struct ept{
        void            *from;
        int             from_id;
        void            *to;
        int             to_id;
        ELEMENT_DATA_TYPE *data;
        struct ept      *next; };
struct ppt{
        int         top, left;
        int     bottom, right;
        PAR_DATA_TYPE   *data;
        struct ppt      *next; };
```

   点と括弧はラベル名通りの値が入っている。枝は始点と終点の部品の座標を示すデータのあるアドレスと、その識別子(点・左括弧・右括弧)を格納している。

3. 部品固有のデータ

   3番目のデータ型は、検索の際に必要なその他のデータで、他の部品との関係に影響されないデータを格納する。特にこの部分のデータ型はDUO表現にも共通で存在する項目が多く、要素については点も枝も同じ型を使用している。

4.1.2 DUO表現

要素・括弧・問い合わせグラフにはそれぞれ、element_type、parenthesis_ type、qg_typeが宣言されている。また要素・括弧の列を表すためのELMS_type、変数名のリストを格納するためのVAR_typeが特別に宣言されている。

```
struct Et{
      int         type_ID;
      void        *parts;
      struct Et   *next;    } ELMS_type;
struct Vt{
      char        *name;
      struct Vt   *next;    } VAR_type;
```

4.2 処理の流れ

4.2.1 グラフ情報→DUO表現変換

グラフ情報からDUO表現への変換はパスをたどりながら行う。パスをたどって行くにあたり、同じパスを2度以上たどらないように部品の変換が済んだかどうかのフラグを立てて行く。

変換の順序は以下の通りに行う。

- 記述にエラーが無いか調べる。
  記述の間違いとなる条件を以下に示す。
  - 要素の点が一つも存在しない。
  - 括弧どうしが交差している。
  - 括弧の片側(左右)の外側(あるいは内側)に点と枝両方が存在している。
  - 括弧の片側の内と外両方に、同じ種類の要素が存在する。
  - 点どうし、枝どうしがつながってパスになっている。

  ただし、最後の条件はインターフェイスでの入力の時点で検査しているため、実際ここの部分では行っていない。

- 全ての部品の座標以外のデータを変換する
- 括弧の中のパスを変換する
  括弧の内部のパスを変換する場合、まず括弧自身を処理済みの状態にする。これにより、処理が括弧の外へ洩れてしまうことを防ぐ。また、括弧の内部に別の括弧が存在するならば、その括弧内のパスを先に変換する。これにより外側の括弧内のパスの変換の際に、内側の括弧内のパスも変換してしまうことを防ぐ。
- どの括弧にも含まれていない点からパスをたどって変換
  パスは点に当たる部分で2つに分けてしまっても、問い合わせを作る際に何の問題も生じない。そのため、変換を開始する点は、括弧の外部にある点ならば、どの点でも良いことが分かる。
- 変換を行っていない括弧を変換
  まず括弧の左に枝が無いかを調べ、存在した場合その枝からまず変換を行う。これは括弧の左どうしが枝でつながっていた場合その枝が処理されなくなってしまうのを防ぐ。先に変換されたパスは逆転され、その最後に括弧から右のパスを連結する。

主な関数は以下の通りである。

1. 点からの変換 conv_node()
   点に接触している部品は枝または括弧 (左右) である。枝と左括弧に関しては、まだ変換を行っていなければ変換を行う。しかし、右括弧に関しては接触していてもそれは無視する。これは括弧の内部を左から変換しており、右からパスをたどって行くには、括弧内のパスを逆転しなくてはならない。逆転は可能だが、実際この時点でパスを切ってしまったとしても何ら問題は無い (意味が変わってしまうことは無い) ためこの様にすることとした。
2. 枝からの変換 conv_edge()
   枝は点または括弧 (左右) に接する。必ず枝の前後に1つずつ枝以外の部品が接しているため、引数に関数の呼び出し元が変換した部品のデータのあるアドレスと識別子が必要になる。また、枝は後の変換で端となっていると変換できないため、たとえ先の部品が変換済みでも、その部品の変換を行う。
3. 左括弧からの変換 conv_par_l()
   内部はすでに変換されているため、対になる右括弧に接する部品の変換を次に行う。
4. 右括弧からの変換 conv_par_r()
   次の変換へ移る前に括弧の内部のパスの逆転を行う。その後、左括弧に接触する部品の変換を行う。ここまでの変換関数は、自分以降のパスのデータを自分のデータの後につなぎ、そのデータ構造のアドレスを返す。
5. 座標以外のデータの変換
   各部品毎に関数が存在する。関数は座標以外のデータを受け取り、DUO 表現の構造体にデータを格納し、そのアドレスを返す。
6. パスの逆転 reverse_E()
   部品列の逆転を行う。逆転のアルゴリズムは、パスの先頭以外を再帰呼び出しで逆転する、返って来たパスの最後に残した部品をつなぐ、そのパスの先頭アドレスを返す、となっている。残すべき部品が、部品列・括弧であった場合、こちらも再帰呼び出しにより逆転させる。

4.2.2 DUO 表現→ Quixote 問い合わせ変換

DUO 表現→ Quixote 問い合わせ変換は、DUO 表現がツリー構造をしているため、関数を再帰的に呼出し、変換を行っている。

各関数の戻り値に必要なものは、変換された Quixote の問い合わせ表現 (以下、問い合わせ表現) と宣言すべき新しいルール (以下、追加ルール) の2つである。どちらも文字列のため、戻り値はその文字列の格納されたバッファの先頭アドレスとなる。このバッファは呼出し元が確保し、引数としてそのアドレスを渡している。

また枝や括弧の変換では、前後の要素の点から変数を借りて来なくてはならない。そこで関数を呼び出すに際に、前後の部品 (要素だけでなく括弧) も引数として渡す。渡す部品が存在しなければ、NULL ポインタを引数とする。

主な関数は以下の4つである。

1. 部品列の変換 ch_ELMS()
   型 ELMS_type に格納された部品列の変換を行

う関数である。この部品列から得られる答えは、格納された部品一つ一つから得られる答えの積集合 (AND) に等しい。そこで関数では大まかに次のような変換を行っている。

- 部品列の先頭から変換を行う。
- 得られた問い合わせ表現をバッファに格納する
- 次の部品の変換を行い、「,」で区切り、バッファの最後へ追加する。
- それを繰り返し、最後の部品まで変換を行う。

また、ここでは追加のルールは発行されないが、子の変換により追加のルールが生成される可能性がある。

2. **要素の変換** ch_elm()
型 element_type に格納されたデータに対して変換を行う関数である。この型に格納されたデータは子のデータを持たないため、再帰的に他の関数を呼び出す必要が無い。そのため、要素から問い合わせ表現を生成する場合、その要素が点であった場合問題無く変換は行われる。しかし、枝であった場合前後の部品から変数名を借りて来なくてはならない (枝自身が持つ変数名は枝自体が持つ値のために割り振られるため)。

3. **括弧の変換** ch_par()
型 parenthesis_type のデータの変換を行う関数である。

この型では、括弧の内部に存在するパスを部品列と同様 ELMS_type の構造体に格納して持っている。しかし、この部品列から得られる答えは前述の部品列の場合と異なり、格納された部品から得られる答えの和集合 (OR) と等しくなる。そのため、まず先に仮の関数名を準備しておき、各部品から得られた問い合わせ表現を追加ルールにその関数名として格納する (= OR の定義)。

また、括弧では再帰の指定もルールに登録する必要がある (その必要が無かった場合は、前記の関数名を問い合わせ表現として格納してやれば良い)。先程とは別の関数名 (説明のため先に準備された関数名をｆ１、後に準備された方をｆ２とする) を準備しておく。もし付された記号が「＊」の場合、まず先に

```
f2(l1=V,l2=V);; (再帰回数０の記述)
```

と追加ルールに記述しておく。続けて、

```
f2(l1=V1,l2=V2) <=f1(l1=V1,l2=V2);;
                        (再帰回数１の記述)
f2(l1=V1,l2=V2) <=
   f1(l1=V1,l2=V),f2(l1=V,l2=V2);;
                        (再帰回数２以上の記述)
```

と記述することによりルールが完成する。後は問い合わせ表現として f2 を格納して終りである。

また括弧でも、要素の枝と同様に変数名を前後の部品から得る必要がある場合がある。

4. **変数名を得る** getvar()
要素の枝や、括弧の所で述べた通り、その部品自体からは変数名が導き出されないことがある。ゆえにこの関数では引数として、その部品のデータ以外に、変数の必要な場所と前後の部品のデータを必要としている。

処理は次の手順で行う。

- もし求めている変数名がその部品自身の値に関する変数名ならば、その部品の持っている変数名を返す。
- もし求めている変数名が点の前後のものである場合、その点の変数名を返す。
- もし求めている変数名が、枝の前 (又は後) のものである場合、その前の部品の後 (前) の変数名を得るように再帰的に関数を呼び出す。
- もし求めている変数名が、括弧の前 (後) のものである場合、括弧の内部のパスの先頭 (最後) の前 (後) の変数名を求めるように再帰呼び出しを行う。
- 再帰呼び出しを行った場合は、その戻り値をそのまま返す。

5 実行例

実行の様子を図5、図6に示す。ここでは2章で示した「都市の連絡データから『nagoya』の下り方面に存在する都市を求める」例を挙げる。

まず部品の配置を行う。編集モードメニューから「Element」を選ぶ。編集パネルが要素モードに替わるので、点・枝指定メニューから「Node」を選ぶ。続けて、ラベル指定を行う。ここでは「city」と記述する。これで要素の点が置けるようになるので、適当な位置をクリックし、点を置く。同様の手順で、括弧、要素の枝も配置すると図5のようになる。

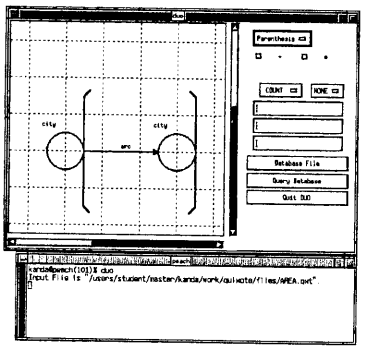

図5 部品の配置

次に、括弧や要素の各部品について詳細の指定を行う。編集モードメニューから「Specify」を選び、すでに配置されている部品をクリックする。ここでは括弧をクリックしたとする。もし、この時に、編集パネル括弧モードでなかった場合、編集パネルが切り替わる。ここで括弧に再帰の指定を行いたいので、再帰指定トグルの「＋」をクリックする。これで括弧の指定が完了した。同様にして、2つの点についても詳細の指定を行う。

問い合わせの記述が完了した時点で問い合わせ開始ボタンを押す。するとインタプリタは問い合わせを行い、結果を出力する (図6)。最後に「終了ボタン」を押し、インタプリタの実行を終了する。

6 考察

6.1 インターフェイス

工夫した点を以下に示す。

- **編集パネルによる編集制限**
  初心者にとって不必要なボタン類は、操作を混乱させる原因である。本インターフェイスでは、出来る限り、不要なボタン類を隠す手段として、編集パネルの切り替えを行なっている。それにより、ユーザは不必要なボタンに惑わされることなく、編集作業を行なうことが出来る。

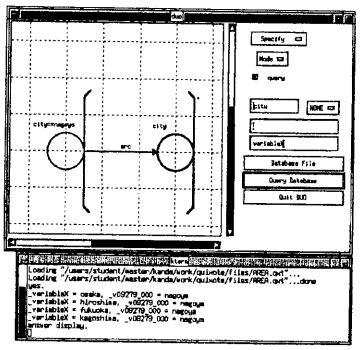

図6 問い合わせ

- **グリッドの設置による記述の曖昧さの解消**
  グラフの描画が自由に出来るのは、見やすいグラフの記述には欠かせない条件である。しかし、制限がなく自由に描画できる場合、その自由さのために、曖昧な記述の氾濫を招くことになる。そこで、グリッドを設置し、部品の配置に制限を設けることにより、様々な記述の曖昧さを解消出来る。

- **詳細指定モードの追加**
  全ての部品に対し、配置前に詳細を指定するのは、非効率的である。詳細指定モードの追加により、同じ部品を連続して配置し、後で一つ一つの部品の詳細を指定することが可能である。

次に使いにくい、または、作業の効率化のため、改良すべき点について示す。

- **結果・スキーマのグラフによる表示**
  本システムでは、結果を、標準出力に出力している。Quixoteの出力をそのまま出力しており、意味はそれなりに分かるが、分かりやすいとは言えない。

- **枝の表示の多角化**
  現在のインタフェイスでは、枝は両端の部品の間を直線で結ぶことしかできない。そのため、部品間に2つ以上の枝が存在する場合枝が重

なってしまう、同じ点を始点・終点とする枝が表示できない、などの現象が起こる。

- **括弧の編集**
  括弧の大きさ、位置の変更が今のインターフェイスでは行なえない。置き直す、コピーするなどにより、その代用は出来るが、スムーズな編集のためには、括弧の編集機能が必要である。

- **パネルへのラベル表示**
  テキスト入力箇所にラベルが付いていないため、このままでは、初心者には使えない。

- **部品群の一括削除・複写**
  複数の部品の一括編集の機能である。この機能の追加のために、新しい編集モードの追加が必要と考えられる。

6.2 変換アルゴリズム

変換アルゴリズムの効率によって全体の性能が左右されるため、インタプリタにとって最も重要である。

まず、問い合わせを表現するグラフ→ DUO 表現変換部について述べる。工夫した点を以下に示す。

- **変換開始前に記述の間違いを探す**
  変換の前に記述の間違いを調べることにより、即座に記述間違いの返答が出来る。

- **先に括弧の内部にあるパスの変換を行なう**
  括弧の内部のパスを先に変換することにより、括弧の外から括弧の内部の点へつながるパスの変換の際に、重複して変換してしまうことを防ぐのが容易になる。

- **点の左側に括弧が接していても変換を行なわない**
  無駄にパスの反転を行なうのを防ぐことにより、効率化を図っている。

また、改善すべき点を以下に示す。

- **変換開始点の決定法**
  パスをたどる変換を行なうに当り、開始点がパスの端にあれば、次の DUO 表現からの変換回数が 1 回少なく済む。また、括弧の変換を左からできる場合、左から行なった方が、右から行うよりも早く変換が終る。本システムで採用している方法は、判定にかかる時間は短く済むが、その点から見ると効率が良いとは言えない。

DUO 表現→ Quixote 問い合わせ変換部については、DUO 表現の時点でほぼ問い合わせの大まかな形は出来上がっているため、行なっていることは構造体に格納されているデータをテキストにする程度の内容である。効率が問題となる部分があまり無い。

7 おわりに

グラフィカル問い合わせ言語 DUO 用のインタプリタとしてデータファイルの選択から結果の出力まで一通り動作するものを実現した。本研究では、実装にあたり、問い合わせに素早い応答をするアルゴリズム、ならびに、使いやすいインターフェイスの 2 点に配慮した。インターフェイスは、編集パネルによる編集制限、グリッドの設置による記述の曖昧さの解消、詳細指定モードの追加により、使いやすいものとすることができた。変換アルゴリズムは、特に問い合わせを表現するグラフから DUO 表現への変換において、変換前に記述間違いチェック、括弧内パスの先行変換、点の左に接する括弧の無視により、効率良いものが実現できた。

今後は、スキーマや検索結果のグラフによる表示、更に使いやすいインターフェイスの設計、より高機能な DBMS への対応などが課題である。

参考文献

[1]. 宝珍輝尚:「グラフィカル問い合わせ言語 DUO の問い合わせ能力」, 情報処理学会論文誌,Vol.36, No.4,pp.959-970(1995).

[2]. 兜木昭男, 木下凌一, 栄谷政己, 林秀幸, 安川悦子:「X-window OSF/Motif プログラミング」, 日刊工業新聞社 (1990).

[3]. D.A. ヤング:「XToolkit プログラミング OSF/Motif 版」, トッパン (1991).

[4]. 井門俊治:「X-Window 実用グラフィックス入門」, 日刊工業新聞社 (1992).

[5]. 財団法人　新世代コンピュータ技術開発機構:「big-Quixote システム利用マニュアル」 (1995).

[6]. 財団法人　新世代コンピュータ技術開発機構:「Quixote 言語マニュアル」 (1995).

[7]. 穂鷹良介:「データベース入門」, オーム社 (1978).

[8]. 中島秀之:「Prolog」, 産業図書 (1983).