データベースシステム 99−19
（1994. 7. 21）

micro-*Quixote* の実現とその拡張機能

新部 裕　　高橋 千恵　　横田 一正
(株) 三菱総合研究所　(財) 日本情報処理開発協会　(財) 新世代コンピュータ技術開発機構

micro-*Quixote* は演繹オブジェクト指向データベース (DOOD) システム *Quixote* の簡易版の実現である. *Quixote* の持つ多くの機能のうち, 問合せ機能として重要で実現が容易と思われる機能を取り出し, C 言語で実装した. micro-*Quixote* では, 包摂関係制約, 属性継承, および仮説生成の機能を利用することができる. また, 外部呼び出しによる拡張機能を利用して, 既存のシステムとともに用いることができる.

本論文では, micro-*Quixote* の実現を述べ, 例題プログラムにより, その機能の実際の利用について示す.

Design and Implementation of micro-*Quixote* and Its Extension Function

Yutaka Niibe　　Chie Takahashi　　Kazumasa Yokota
Mitsubishi Research Institute, Inc.　　Japan Information Processing Development Center　　Institute for New Generation Computer Technology

Micro-*Quixote* is a simplified implementation in C language of the deductive object-oriented database *Quixote*. Micro-*Quixote* has features of subsumption constraints, property inheritance, and hypothetical generation, which can be considered important as advanced query processing and easy to be implemented. Further, micro-*Quixote* has an extended features, external invocation, which makes cooperation with other systems possible.

In this paper, we describe the features of micro-*Quixote* and their implementation, and show its usefulness by an example.

1 はじめに

micro-*Quixote* は, 演繹オブジェクト指向データベース (DOOD) システム *Quixote* の機能の一部を取り出し, C 言語で実装した小さいシステムであり, UNIX, MS-DOS, および Macintosh などの環境で稼働する[1].

データベース・システムをファイルシステム, 狭義の DBMS, 問合せ言語の 3 層と考えた時, *Quixote* は DBMS と問合せ言語を実現しており, ファイルシステムに関しては UNIX ファイルシステムや Kappa DBMS [2] とのインターフェースを持っている. *Quixote* の 問合せ言語の面を考えた場合, その問合せ機能は強力であり, 演繹部分だけを考えても, 関係代数の拡張である Datalog¬ [3] と上位互換性を持っている. micro-*Quixote* はこの *Quixote* の問合せ機能に着目し, その一部分の簡易な実現を行なったものである.

問合せ機能に着目したのは, *Quixote* の問合せ機能の部分を, もっと気軽に一般の DBMS とつないで楽しみたいという要求が多くあったためである. *Quixote* には遺伝子情報処理, 法的推論などの応用があるが [2], そのすべてを *Quixote* システムで実現するのではなく, 既存の遺伝子や法律などのデータベースを利用したいという要求があった. また, データベース以外の既存の計算機構を用いたいという要求もあった. *Quixote* は異種分散処理環境 Helios の枠組において, これらの要求に答える予定である [6]. micro-*Quixote* は Helios の実現に先だって, この要求に対する一つの解を示すことを狙い, 拡張機能を実現している.

本論文では, micro-*Quixote* の実現と拡張機能について述べる. 2 節では micro-*Quixote* の設計方針を述べる. 3 節では micro-*Quixote* の実現の詳細について述べる. 4 節では具体的に micro-*Quixote* の例題のプログラムにより利用例を紹介し, 実現した機能がどのように用いられるかを示す.

2 設計方針

micro-*Quixote* の開発では以下の二つの点を大きな方針とした.

1. *Quixote* の機能のうち容易に実装可能な中心的機能を実現する. 実現に際して開発コストが大きな機能は対象としないか, または単純化して実現する.
2. 簡易に実装する. 開発が容易であることを重視し, 実行速度, メモリ使用効率などは重視しない.

そして, 以下の設計方針を取った.

- システム
  小さく簡潔なシステムとする. スタンドアローンで動くシステムとし, 単一ユーザの利用のみを考える. *Quixote* システムで実現されているネットワーク上で複数のユーザにサービスする, サーバとしての機能は持たせない.
- 利用環境
  メモリ, ディスクなどの資源の乏しい環境でも動くことを考慮する. 移植性を重視し, 多くの環境で利用できることを狙う.
- 言語
  言語は *Quixote* のサブセットとし, わかりやすい必要最小限の文法とする.
- 意味論の変化
  Quixote の機能を単純化して実現した場合, 意味論が変化するが, これは容認する.

3 実現

本節では micro-*Quixote* の実現について, *Quixote* の実現との違いを中心に述べ, なぜ, その実現を用いたのかについて説明する.

micro-*Quixote* では, *Quixote* の機能のうち, 包摂関係制約, 属性継承, および仮説生成を実現している. 独自の機能として外部呼び出しを付加し, micro-*Quixote* を拡張する機能としている.

3.1 micro-*Quixote* のシステム構成

micro-*Quixote* のシステムは図 1に示すように, 推論機構, 制約解消系, 外部呼び出しインターフェース, 属性継承部, 入出力 (parser/printer), および包摂関係 / ルール管理部の 6 つの部分から構成される. 以下ではこの各部のうち, 推論機構, 制約解消系, 外部呼び出しインターフェース, および属性継承部について述べる.

[1] *Quixote* システムは, ICOT で開発された並列推論マシン PIM 上の並列論理プログラミング言語 KL1 で実装されており, 現在, UNIX 環境に移植中である [1].

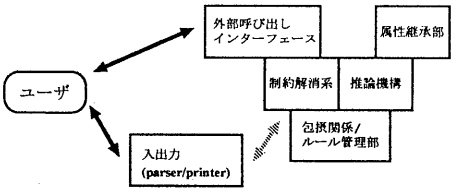

図 1: micro-*Quixote* のシステム構成

3.2 推論機構

micro-*Quixote* の計算ルールは Prolog の計算ルールと同じく, 登録されたルールを上から下, サブゴールを左から右に展開して計算を進めることとした. これは *Quixote* の計算ルールとは全く異なっている. *Quixote* では OID による属性のマージ, およびデータベースとしての全解探索と停止性の保証のために幅優先で計算を行なっている. しかし, この *Quixote* の計算ルールをそのまま実現するには開発コスト, 実行時の効率の問題があると判断し, micro-*Quixote* では Prolog の計算ルールを採用した.

この実現の違いは micro-*Quixote* と *Quixote* でプログラムの意味を変えてしまう. *Quixote* では同一の OID による属性が AND の関係にあり, マージされていたが (以下この機能をマージと略す), micro-*Quixote* では OR となってしまう. つまり, micro-*Quixote* では *Quixote* における OID の意味論が実現されていない.

仮説生成は推論機構で行なわれるが, この実装は *Quixote* とほぼ同じである [4].

3.3 制約解消系

micro-*Quixote* の制約には, オブジェクトの属性にかかる制約, 包摂関係制約, および外部呼び出し制約の 3 種類があり, それぞれ対応した制約解消系により処理される. ただし, micro-*Quixote* では属性制約に集合制約は含めなかった.

制約は種類に応じて, 別々のリストとして管理される. *Quixote* の制約解消系は, 変数と変数にかかる制約という形で制約に関するデータを管理するが, micro-*Quixote* では制約を言語上の表現と対応させた形式で実現する. *Quixote* の実現は実行効率に勝る. micro-*Quixote* では実行効率ではなく, 簡潔な実現を目標においたため, この実現を選択した.

3.4 属性継承部

図 1で示したように, 属性継承は, 推論機構と関連して実現される. この実現は *Quixote* の実現とほぼ同じである [5]. しかし, 3.2節で示した推論機構の違いにより, 解に違いが出てくる. つまり, micro-*Quixote* は深さ優先の計算ルールのために, 制約の比較やマージは行なっておらず, それらがすべて列挙される. したがって, *Quixote* ではマージして得られる解は micro-*Quixote* が列挙する解の中の一つとして出力されるだけであり, micro-*Quixote* は無駄な解をたくさん見せてしまう. また, *Quixote* でマージにより実行時にエラーとして検出される矛盾した属性は, micro-*Quixote* では検出されず, 一つの解として示されてしまう.

3.5 外部呼び出しインターフェース

micro-*Quixote* は機能を拡張する仕組みとして, 外部呼び出しを導入した. 図 1で示したように, 外部呼び出しは, 制約解消系と関連するものとして実現される. これは, 外部呼び出しを制約として実現することが言語として導入が容易であったためである.

外部呼び出しの実現では, 2節で述べた方針に基づき, micro-*Quixote* には最小限の機構を用意するのみとし, micro-*Quixote* の外部になんらかのディスパッチャが存在することを仮定している. このディスパッチャは micro-*Quixote* からの計算の要求を適切に振り分けて, 計算をさせるという役割を持つ. この micro-*Quixote* とディスパッチャの切り分けにより, 機能の拡張は micro-*Quixote* 本体の変更を全く必要とせず実現できる.

制約解消系と外部呼び出しインターフェースは, 以下のような動きをする.

1. 計算中に外部呼び出し制約がバインドフックにあたった時に, 外部呼び出しインターフェースに制御が渡る
2. 外部呼び出しインターフェースはその制約を外部に出力して, 答を待って計算を中断する
3. ディスパッチャが計算を振り分け, 外部計算機構に計算させて答を得る
4. 外部呼び出しインターフェースは, ディスパッチャより返ってきた答を取り込み, 制約解消系に渡す.

現在は外部のディスパッチャとして, Emacs を用いたものが実現されている. 図 2にその様子を示す. ここで

特徴的な点は, ユーザインターフェースの役割と, ディスパッチャの役割を重複させてEmacs に持たせている点である. この仕組みでは, Emacs を中心としてユーザ, 外部計算機構, および micro-*QUIXOTE* が対称となっており, micro-*QUIXOTE* から外部計算機構を利用することだけでなく, 外部計算機構から micro-*QUIXOTE* を利用することも可能としている.

ディスパッチャと micro-*QUIXOTE* のやりとりで, 外部呼び出しの計算は以下のように進む.

1. ユーザから問合せが出される
2. その問合せをユーザインターフェースとしての Emacs が micro-*QUIXOTE* に伝える
3. micro-*QUIXOTE* が計算を実行する
4. micro-*QUIXOTE* から外部呼び出しが行なわれる
5. micro-*QUIXOTE* の外部呼び出しをディスパッチャとしての Emacs が検出し, ディスパッチする
6. ディスパッチした結果が戻ってくる
7. Emacs より micro-*QUIXOTE* に結果が送られる
8. micro-*QUIXOTE* は結果を使って計算を続行する
9. 4 に戻るか次に進む
10. Query の結果が求まる

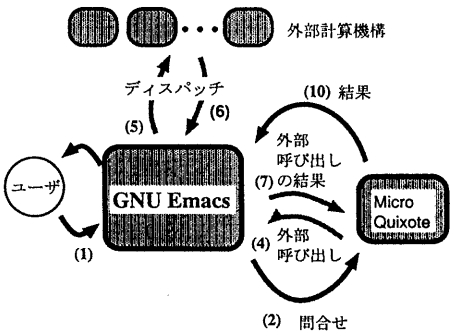

図 2: Emacs による外部呼び出しディスパッチャ

4 micro-*QUIXOTE* の使用例

本節では, micro-*QUIXOTE* の実際の使用例を示し, 3 節で述べた機能がどのように用いられるかを示す. まず, micro-*QUIXOTE* のプログラムを示し, それについて解説をする. 次にそれに対する, 問合せの例を示す.

4.1 プログラムの例

例題プログラムは micro-*QUIXOTE* から ftp の外部呼び出しを行ない, ソフトウェアの検索を行なうプログラムである. このプログラムは, 外部のデータベースの利用に際して, 実際の問合せの時点で情報を取得するという方針に基づいて書かれている.

4.1.1 プログラムの構成

micro-*QUIXOTE* のプログラムは二つのパートからなる. 一つは包摂関係の定義で, もう一つはルールの記述である. ルールはモジュールという単位で記述することが可能である[2]. 例題プログラムは以下のような構成となっている.

```
&program;;
&subsumption;;  % 包摂関係定義の始まりを示す
   % オブジェクト間の包摂関係定義
&rule;;   % ルールの始まりを示す
   % ftp サイトに関するモジュール
   % エディタに関するモジュール
   % ソフトウェア配布に関するモジュール
   % ファイルに関するモジュール
   % 外部呼び出しに関するモジュール
&end.
```

このように, micro-*QUIXOTE* のプログラムはキーワード `&program` で始まり, `&end` で終る. キーワード `&subsumption` から包摂関係の定義が記述され, キーワード `&rule` からルールが記述される. ここでは, ルールは 5 つのモジュールで構成されている.

以下, プログラムの各部分について述べる.

4.1.2 オブジェクト間の包摂関係定義

ここでは, オブジェクト間の包摂関係を定義する.

包摂関係 (=<) の左辺は, 右辺に包摂されることを意味する. 例えば, `gcc =< software` は `gcc` は `software` であることを表現している. 以下のプログラムでは, Emacs と GNU, およびソフトウェアの配送メディアについてそれらの関係を表現している.

[2]現在の micro-*QUIXOTE* ではモジュールの機能は外部のフィルタとの組合せで実現されており, micro-*QUIXOTE* 本体では実現されていない. 将来この機能は micro-*QUIXOTE* 本体に取り込む予定である.

```
% gcc はソフトウェアである
gcc =< software;;
% エディタはソフトウェアである, ツールでもある
editor =< {software, tool};;
% Emacs はエディタである
emacs =< editor;;
% Emacs, GCC は GNU の product である
gnu >= {emacs, gcc};;
% Mule は Emacs を特化したものである
mule =< emacs;;
mule >= { "mule-1", "mule-2"};;
% media には, カートリッジ, 8mm テープ,
% フロッピーがある
media >= {"CMT", 8mmTape, floppy };;
```

この関係は図示すると図3のようになる (包摂関係は束を構成する (図中 ⊤ と ⊥ は省略している)).

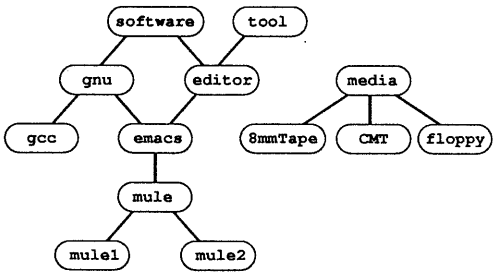

図 3: 包摂関係から作られる束

4.1.3 ftp サイト

このモジュールでは ftp サイトについての知識を表現している. ftp サイトはなんらかのカテゴリに基づいてファイルを集めており, 場所を示すドメイン名とディレクトリを示すパス名で特定されるものであるとし, それらをオブジェクトで表現している.

```
ftp:: % ftp サイトのリスト
{
   % etlport には mule がある
   ftp_site[category=mule,
            where="etlport.etl.go.jp",
            directory="/pub/mule"];;
   % prep には gnu がある
   ftp_site[category=gnu,
            where="prep.ai.mit.edu",
            directory="/pub/gnu"];;
}
```

Quixote のオブジェクトはこのようにラベルとその属性の組を持ったものとして表現される. 例えば, 最初のオブジェクトは, mule というカテゴリでファイルを集めている ftp サイトを, category というラベルに対して mule という属性を持つ (ftp_site) として表現している.

4.1.4 エディタに関しての記述

このモジュールでは, エディタに関しての知識を表現している. エディタに関するオブジェクトとその属性を記述している. また, 配送メディアに関するルールも記述している.

```
editor:: % エディタに関しての知識 (mule と emacs)
{   % emacs の拡張言語と使用するメモリ
    emacs/[ext_language="Emacs Lisp",
           required_memory="Big"];;
    % mule は多国語の text を扱う
    mule/[text=multi_lingual];;
    % mule-1 は emacs-18 をベースとしている
    "mule-1"[based="emacs-18"]
           /[status=released,size="Big"];;
    % mule-2 は emacs-19 をベースとしている
    "mule-2"[based="emacs-19"]
           /[status=alpha,size="Very Big"];;
    % ソフトウェアの配送は
    Software/[distribution=D]
     |{Software =< software} <=
      distribution:distributable  % 配送形態と
       [software=Software, media=M, form=FORM],
      distribution:D/[media=M]  % メディアにより
      || { D =< FORM };;          % 決定される.
}
```

このように *Quixote* では属性をオブジェクト固有のものと非固有のものの二種類にわけて, 固有の属性を / の左側に非固有の属性を右側に書く. 例えば,

```
"mule-1"[based="emacs-18.59"]
        /[status=released,size="Big"]
```

は固有の属性として based を持ち, 非固有の属性として status, size を持つ mule-1 を表現している. 非固有の属性は, 包摂関係に基づき継承されるため, 4.1.2節の定義に基づき, emacs の属性は mule および mule-1, mule-2 と継承される.

このモジュールの最後のルールは, 頭部と頭部制約, およびボディとボディ制約という *Quixote* のルールの一般的な形となっている. 頭部とボディは <= で区切られ, 頭部制約は | の後に, ボディ制約は || の後に書かれる.

4.1.5 ソフトウェア配布

このモジュールではソフトウェア配布についての知識を表現している. ソフトウェアはライセンスと大きさによって

配布形態が決まること, 配布メディアにはどのようなものがあるか, 配布形態にはどのようなものがあるかに関して記述を行なっている.

```
distribution:: % ソフトウェア配布に関する知識
{ % ソフトウェアのメディアと形態は
   distributable[software=Software,
                 media=M,form=FORM]
    <= Software/[ % ソフトウェアのライセンスと
        licence="GNU General Public Licence",
        size=SIZE], % サイズで決まる.
       distribution:M/[size=SIZE]
        || { Software =< software,
             M =< media,
             FORM=freely_distributable };;
   % メディアとそのサイズ
   floppy/[size=small];;
   "CMT"/[size="Big"];;
   8mmTape/[size="Very Big"];;
   % 配布形態とメディア
   freely_distributable[media=floppy];;
   freely_distributable[media="CMT"];;
   freely_distributable[media=8mmTape];;
   charged[media=floppy];;
}
```

4.1.6 ファイル

このモジュールでは, 外部のデータベースと micro-*QUIXOTE* のオブジェクトを結びつけている. micro-*QUIXOTE* のプログラムには直接記述されないオブジェクトを, あたかもあるかのように見せる. この micro-*QUIXOTE* のオブジェクトと外部のデータベースの結び付きは, 解の探索時に動的に行なわれる.

```
file:: % micro-QUIXOTE のオブジェクトを
{       % ftp サイトのファイルに対応させる
   FILE/[category=C,where=Where] <=
     ftp:ftp_site[category=C,where=Where,
                  directory=P],
     ext:ftp_ls[where=Where,directory=P,
                file=FILE],
     ext:ftp_file_date[file=FILE],
     ext:ftp_file_owner[file=FILE];;
}
```

4.1.7 外部呼び出し

このモジュールは他のモジュールとは趣を異にする. このモジュールで実現するのは, オブジェクトの表現ではなく, 外部呼び出しの手順である.

```
ext:: % ftp_ls, ftp_file_date, ftp_file_owner
      % の外部呼び出し
{ % file[where=Where,what=What]
  %   /[owner=Owner,date=Date]
  % ftp サイトに対し ls を行なう手順.
  ftp_ls[where=Where,directory=D,file=F] <=
   &true ||          % 繰り返しを行なう.
     { F ## ftp_ls[site=Where,directory=D],
       F =< &top[where=Where,what=What] };;
  ftp_ls[where=Where,directory=D,file=F] <=
  ftp_ls[where=Where,directory=D,file=F] ||
     { F ## ftp_ls[site=Where,directory=D],
       F =< &top[where=Where,what=What] };;
  % ファイルの日時と所有者の情報を取得する
  ftp_file_date[file=FILE]
    |{ FILE.date == Date } <= &true ||
     { Date ## ftp_file_date[file=FILE] };;
  ftp_file_owner[file=FILE]
    | { FILE.owner == 0 } <= &true ||
     { Owner ## ftp_file_owner[file=FILE] };;
}
```

4.2 問合せの例

本節では, 例題プログラムに対する問合せとして, ファイルの検索と emacs に関する情報の取得の例を示す. 以下では下線が引かれている部分がユーザの入力である.

まず, Emacs に関するファイルを列挙したいとする. このために, モジュール file を利用し, カテゴリ emacs でサービスされている FILE を問合せる.

```
?- file:FILE/[category=emacs].
no.
```

そのようなファイルは見つからず, no と答が返ってくる. これはカテゴリ emacs で直接サービスしている ftp サイトがないためである. これに対して二通りの方法で検索条件を広げる.

まず, カテゴリに対する検索条件を上位概念を含めて検索するようにすることによって, 検索する ftp サイトの対象を広げることを意図する. また, emacs だけを対象としたいので, ファイルが何であるかという属性の what で emacs を指定する. この探索において, ftp サイトの探索の対象は emacs より上位の概念となり, 束の上で図示すると図 4 の網かけをした部分となる.

```
?- file:FILE/[category<-emacs,what=emacs].
FILE = "emacs-18.59"[where="prep.ai.mit.edu",
                     what=emacs]
FILE = "emacs-19.25"[where="prep.ai.mit.edu",
                     what=emacs]
```

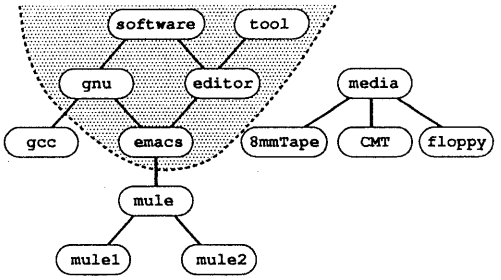

図 4: emacs より上の探索

emacs の二つのバージョンが見つかった. これは, カテゴリに emacs の上位概念である gnu があたって, gnu をサービスする ftp サイト, "prep.ai.mit.edu" で emacs が検索されたためである.

次に, category=emacs という指定が一般的過ぎて見つからなかった可能性を考慮して, 今度はカテゴリに関する検索条件をより具体的にし, emacs より下位概念と指定して聞いてみる. この探索において ftp サイトの探索の対象は emacs より下の概念であり, 束の上で図示すると図5の網かけをした部分となる.

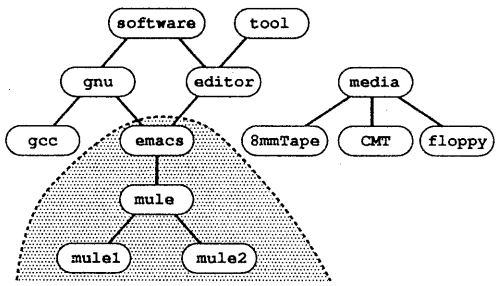

図 5: emacs より下の探索

```
?- file:FILE/[category->emacs].
FILE = "mule-1.0"[where="etlport.etl.go.jp",
                  what=editor:mule]
FILE = "patch-1.0-01"[where="etlport.etl.go.jp",
                      what=patch]
FILE = "README-1.0"[where="etlport.etl.go.jp",
                    what="README"]
```

mule, patch, README が見つかる. これは, emacs の下位概念である mule をサービスする ftp サイト "etlport.etl.go.jp" でファイルが検索されたからである. 属性の what を特に指定しなかったため, patch や README も解となっている.

上記の検索では, 外部呼び出しで, ftp が起動され, micro-*Quixote* は ftp サイトから動的にデータを取得している. この外部呼び出しの様子を図示すると図 6 のようになる.

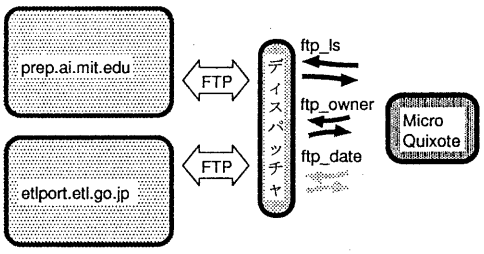

図 6: 外部呼び出し

上の問合せの結果に, editor:mule というのが出てきた. これは mule に対応するオブジェクトである. そこで, mule に関して質問をしてみる. まず, そのスキーマを得る.

```
&> &show_labels mule;;
labels for mule:
        based
        distribution
        ext_language
        required_memory
        size
        status
        text
```

ext_language, status などの属性を持つことがわかる.

その中の拡張言語について指定して, 問合せてみる.

```
?- editor:mule/[ext_language=EL].
EL = Unbound if { editor:mule.ext_language==EL }
EL = Unbound { editor:mule.ext_language==EL,
               "Emacs Lisp">=EL }
```

すると二つの解が示される. 最初の解は, 問合せそのままであるが, これは 3.4節で述べた通り, micro-*Quixote* は属性継承の実現を可能性の列挙としているためである. 二番目の解により, mule の拡張言語は, Emacs Lisp に包摂されるものだとわかる.

この問合せに対する答えは属性継承の機能により得られている. mule は emacs の下位概念であるという包摂関係の定義により, ext_language, memory の属性が emacs から継承される.

さらに, mule に関しての質問を続ける. mule という概念に包摂されるオブジェクトをモジュール editor の中で, 列挙してみる. この問合せには包摂関係の制約を用いる.

```
?- editor:MULE|{MULE =< mule}.
MULE = mule
MULE = "mule-1"[based="emacs-18"]
MULE = "mule-2"[based="emacs-19"]
```

すると概念としての mule, 具体的なバージョンの mule-1 と mule-2 とがあることがわかる. mule-1 は mule-1 に対応したオブジェクトで, mule-2 は mule-2 に対応したオブジェクトであるが, ここで, mule-1 は emacs-18 をベースとしているという固有の属性を持っていること, 同様に mule-2 は emacs-19 をベースとしているという固有の属性を持っていることがわかる.

先の質問により, mule-2 というオブジェクトを得たので, その属性の distribution について問合せる.

```
?- editor:"mule-2"[based="emacs-19.25"]
                  /[distribution=D].
D = freely_distributable[media=8mmTape]
    if { "mule-2"[based="emacs-19"].licence==
                  "GNU General Public Licence" }
```

すると, mule-2 のライセンスが "GNU General Public Licence" であったなら, (8mm テープで) 自由に配布可能であるという, 仮説のついた答が返ってくる. これは mule-2 のライセンスについて情報がデータベース中にないためである.

5 おわりに

micro-*QUIXOTE* は簡易の実装ではあるが, 4 節で示した通り, *QUIXOTE* の問合せを行ないたいという場合には, 十分利用することができる. 一番の問題点は計算ルールの違いによるものであるが, これは 2節の設計方針を変更しなければ実現困難である.

micro-*QUIXOTE* 本体の今後としては, 以下のことを検討している.

- モジュールの機能を本体として実装する.
- ユーザの利便のために, オブジェクトやラベルに付ける注釈をデータとして扱う機能を付加する.

外部呼び出しによる拡張機能については, micro-*QUIXOTE* の実現方式は一般的なもので他のシステムにも用いることができる. これは, 処理系本体には手を加えず機能拡張ができるという特徴があり, 既存のシステムを互いに接続して, 利用することを可能としている. なお, ユーザインターフェースとディスパッチャを重複させるという実現のヒントは複数のインタラクティブなプログラムを処理する expect の処理系から得たものである [8].

現在までに, 拡張機能として実現したものには, 例題で示した ftp によるアクセスの他に, 正規表現による文字列のマッチングと四則演算がある. 今後は Z39.50 プロトコルとのインターフェースを用いて, ネットワーク上のデータベースにアクセスすることを検討している [7]. また, 3.5 節で可能性を示した, 他のアプリケーションから micro-*QUIXOTE* を用いることについても検討を進める予定である.

謝辞

QUIXOTE グループの皆に感謝する. また, 管理工学研究所の堀川勉氏に感謝する. micro-*QUIXOTE* の実現に関する大きな方針, および最初の版の実現は氏による.

参考文献

[1] 津田, 横田, 森田, 高橋, 他: "*QUIXOTE* システム第 3 版", *ICOT Technical Memo TM-1285*, 1993.

[2] K. Yokota and H. Yasukawa, "Towards an Integrated Knowledge-Base Management System — Overview of R&D on Databases and Knowledge-Bases in the FGCS Project," *Proc. FGCS'92*, ICOT, June 1-5, 1992.

[3] Jeffrey D. Ullman, *Principles of Database and Knowledge-Base Systems*, Volume I, Computer Science Press.

[4] 西岡利博, 小島晁, 津田宏, 横田一正, "演繹オブジェクト指向データベース言語 *QUIXOTE* の手続き的意味論", 第 94 回データベースシステム研究会, 7 月, 1993.

[5] 高橋 千恵, 森田 幸伯, 西岡 利博, 津田 宏, "演繹オブジェクト指向データベース言語 *QUIXOTE* の手続き的意味論", 第 94 回データベースシステム研究会, 7 月, 1993.

[6] 横田一正, 相場亮, "マルチエージェントによる異種問題解決系の構想", *MACC'93*, 12 月, 1993.

[7] Brewster Kahle and Art Medlar, "An Information System for Corporate Users: Wide Area Information Servers," *Thinking Machines technical report TMC-199*, April, 1991.

[8] Don Libes, "Kibitz - Connecting Multiple Interactive Programs Together," *Software - Practice & Experience*, Vol. 23, No. 5, May 1993.