2M-7 関数型モデルを用いた化学グラフデータベース CHARM のユーザーインターフェース

山田潤二* 兪玉琴** 大保信夫*** 北川博之*** 山口和紀*** 鈴木功*** 藤原譲***
(*筑波大学理工学研究科 **筑波大学工学研究科 ***筑波大学電子情報工学系)

1．はじめに

近年、化学グラフ構造に対するデータベース化の要求が高まってきているが、化学グラフのような構造を持った複雑なデータをデータベースに格納しようとすると、複数のリレーションにまたがった表現となってしまい、データ処理が難しく効率も悪くなる。これに対して、関係データベースに抽象データ型（ADT）を導入し、その上に関数型のビューを構築することによって、化学者にとって理解しやすい化学グラフのデータベースを作成する。この方式をとることの利点としては（1）システムの内部構造として従来の関係型のDBMSを用いているのでデータの完全性、一貫性、障害回復等の面で安定したデータ操作ができる。しかも、（2）ユーザーは内部の関係型の複雑なスキーマを全く意識することなく関数型のデータ操作ができる。さらに、（3）アプリケーション特有のADTを定義できるので、非常に柔軟なデータ操作が可能である。等が挙げられる。

本稿では、システム全体の中で、プログラム言語Lispとデータベース操作言語を統合した高度なユーザーインターフェースの実現法について述べる。尚、CHARMシステムはUNIX環境上にあるLispベースの商用関係型DBMS,G-BASEを基盤にして構築されている。

2．システム構成

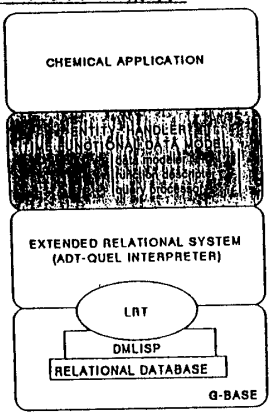

図1 CHARMシステムアーキテクチャ

CHARMシステムのシステム構成について述べる（図1参照）。最下層には前述のG-BASEがあり、その上にADTを扱う能力を持つ拡張関係型システムがあり、さらにその上に関数型のビューを構成するエンティティハンドラが存在し、化学グラフアプリケーションが構築される。

3．ユーザーインターフェース

ユーザーインターフェースはプログラム言語Lispの環境中で関数型のデータベース・ビューを構築することが主目的であり、システム全体構成の中ではエンティティハンドラ層に対応する（図1参照）。エンティティハンドラは、データモデル化部（3.2, 3.3節参照）、関数記述部（3.4節参照）問い合わせ処理部（3.5節参照）から構成される。本システムは以下の点で特徴的である。

- ユーザーインターフェースは全てLispの関数として実現される。
- ユーザーがADTを定義可能である。
- ユーザー定義のADTと基本データ型を用いて関数型のデータベーススキーマ（ビュー）が定義可能である。
- 定義されたスキーマに対してプログラム環境中からデータベース操作が可能である。
- ユーザーが化学グラフオブジェクトの手続き的な意味を記述する複合関数を定義できる。

3．1．ユーザーインターフェース概観

ユーザーは、アプリケーションプログラム中に各インターフェースを取り込むことによって関数型データベースのビューを作り出し、内部ビューである関係データベースのスキーマを全く意識することなく関数型のデータ操作を行なうことができる（図2参照）。

3．2．ADTの定義

スキーマの定義に先立ってユーザーは、必要に応じてADT-definition関数を起動し、システムと対話しながらADTとADTを操作するためのADT関数を定義する。ここで定義されたADTは、内部構造である関係データベースにおいてもADTとして認識される。

```
ADT-definition(
        ADT-name =  TREE
        PARENT   =  nil
        B-function in
                is TREE X list-of TREE ---> boolean
        B-function display
                is TREE ---> boolean

        filename    = "/usr1/ChemDB/adt/tree" )
```

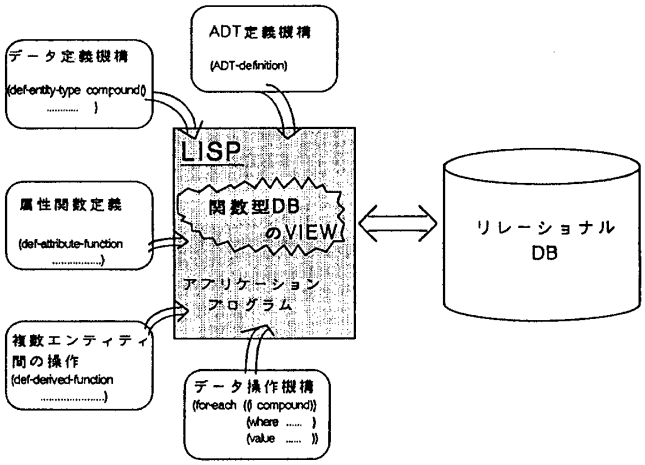

図2 LISP環境からのデータベース操作概略

Chemical Graph Database (CHARM) based on Functional View
Junji Yamada* Yu-qin Luan** Nobuo Ohbo*** Hiroyuki Kitagawa*** Kazunori Yamaguchi*** Isao Suzuki*** and Yuzuru Fujiwara*** : * Division of Scientific Technology, ** Program in Engineering Sciences, and *** Institute of Information Sciences and Electronics Univ. of Tsukuba

上の例はTREE型のADT定義の一部である。この例では、TREE型は親となるADTはなく、リターン値が論理値であるADT関数inとdisplayを持っており、その関数はfilenameで示すファイルに格納されていることを表わしている。

3．3．スキーマ定義

エンティティ型は恒久的なオブジェクトとしてスキーマ定義用マクロ def-entity-type を用いて以下のように定義する。ここで定義されたクラスは多重継承によって複数の親クラスを指定できる。関数型のスキーマが定義されると各エンティティの持つ属性は、内部ビューである関係データベーススキーマの属性にマッピングングされ、その値へは属性関数を用いてのみアクセス可能となる。

```
(def-entity-type compound ()
     ((ID   char)
      (NAME   char)
      (FORMULA   CHEMICAL-FORMULA)
      (NUM-OF-ATOMS   int)
      (NUM-OF-EDGES   int)
      (ATOMS   set-of ATOM)
      (STRUCTURE   CHEMICAL-GRAPH)
      (SEGMENTATION   SC-TREE)
      (COMPONENT   set-of SUBSTRUCTURE))
      "/usr1/ChemDB/adt/ret-func")
```

上の例では、COMPOUNDというエンティティ型が定義されている。ID,NAME等は属性関数であり、リターン値はSC-TREE,SUBSTRUCTURE等エンティティ型でもよい。以下に、マッピングングされた関係型のスキーマの一部を示す。尚、アンダーラインのついたものはADTのフィールドである。

R1:COMPOUND(C#,NAME,FORMULA,CHEMICAL-GRAPH,..)
R2:SUBSTRUCTURE(S#,EDGE,CHEMICAL-GRAPH,.....)
R3:CONNECTION(CG#,EDGE,GRAPH,EDGE-MAPS,.....)
R4:LINK(L#,LINKS)

3．4．データ操作関数定義

各エンティティに付随した属性にアクセスするために属性関数を def-attribute-functionマクロを用いて以下のように定義する。関数の本体はADT-QUELの形式である。この関数を定義することによりデータベース中のデータをスロットとしてアクセスすることが可能となる。

（例）

```
(def-attribute-function ATOMS (i)
   (range of t is COMPOUND
    retrieve(set-of-atoms t.CHEMICAL-FORMULA)
    where t.C# = ID(i)))
```

この属性関数ATOMSはCOMPOUND型のエンティティに対して定義されたものであり、化合物を構成する原子の集合を返す機能を持つ。この中で関数set-of-atomsはCHEMICAL-FORMULA ADT上で定義される、化学グラフを構成する原子のラベルの集合を返す関数である。また、複数エンティティにまたがった操作はdef-derived--function によって以下のように定義する。

（例）
```
(def-derived-function INCLUDE-BLOCK (i)
        (INCLUDE   (SEGMENTATION i)))
```

この例は、COMPOUND型のエンティティ（化合物）に対して、 それに含まれるスーパーブロック[2]の集合を返す関数であり、COMPOUND型、SC-TREE型の２つのエンティティにまたがった操作がなされている。操作は、COMPOUND型の属性関数SEGMENTATIONによって得られたSC-TREE型のエンティティに対してSC-TREE型のエンティティの属性関数INCLUDEを実行することにより実現されている。

3．5．問い合わせ処理

以上のように定義することによりデータベースに対する問い合わせは for-eachスペシャルフォームを用いて以下のように行なえる。for-eachは、恒久オブジェクトをクラスごとに集合として扱い、実行時には指定された変数にそのクラスのインスタンスが１つずつバインドされ、指定された条件節(where節)を満足するオブジェクトに対してのみ values 節を実行する。for-eachインターフェースで受け付けられた問い合わせ文は、ADT-QUELインタープリタ、LRTを通して、低レベルデータ操作機能を持つDM-Lispの形式に変換され、データベースへのアクセスが行なわれる（図１参照）。

（例）

```
(for-each  ((I  SUPER-BLOCK))
  (where (<=  (NUM-OF-EDGES i) 20))
  (values (mapcar 'display
          (mapcar 'STRUCTURE (PARENT i)))))
```

この例は、エッジの数が20以下のスーパーブロックを含んでいる化合物のグラフを全て表示するという問い合わせを示している。NUM-OF-EDGES,PARENTはSUPER-BLOCK型の、STRUCTUREはCOMPOUND型の属性関数であり、displayはCHEMICAL-GRAPH ADTのADT関数である。これと同じ内容の問い合わせを直接関係データベースに対して行なうとすると、複数の関係間のリンクを複雑にたどる操作を記述しなければならず、ユーザーにとって非常にわかりにくく、間違いやすいものとなってしまう。このことからも関数型のビューを用いる利点が浮き彫りとなる。

4．おわりに

現在、以上の方法に基づいて、関数型のビューによるデータベース操作がなされているが、今後は問い合わせの高速化のためのバッファリング法、問い合わせ最適化等の課題に取り組んでいく予定である。

5．参考文献

[1]Yu X.,Ohbo N.,Kitagawa H.,Fujiwara Y., Integration of Functional Data Model into Programming Environment: Application to Solid Database,Proceeding of IFIP TC2 Working Conference on Visual Database Systems,Tokyo (1989).

[2]Yu-qin Luan.,Ohbo N.,Kitagawa H.,Fujiwara Y., Functional Approach to Chemical Structure Database,Proceeding of Int. Symp. on Database Systems for Advanced Applications, Seul(1989).　他