データベースシステム　97－9
（１９９４．　３．　１８）

Ｃ＋＋によるＪＤＭＦ／Ｍｏｄｅｌ－１９９２の実現

堀越貴之，野口宏
茨城大学

１９９２年に、オブジェクト指向を採用したＪＤＭＦ／Ｍｏｄｅｌ－１９９２（以下、ＪＤＭＦ－Ｍ９２）が標準データモデル機能として規定された。ＪＤＭＦ－Ｍ９２はデータ構成規則のみが記述されており、データ操作規則については概念規定のみでその具体的な表現方法及び実装方法は規定されていない。従って、ＪＤＭＦ－Ｍ９２の実現のためにはデータ操作規則を用意する必要がある。

本稿はＪＤＭＦ－Ｍ９２を機種に依存しないＣ＋＋で実現することを目的とし、実現のための方針を決定した。さらに、この方針と規定されたデータ構成規則に従うデータを生成するためのメソッドを用意し、データ構成規則とともに実現を行なった。

キーワード：ＪＤＭＦ，オブジェクト指向，データモデル，Ｃ＋＋

Implementation of JDMF/Model-1992 using C++

Takayuki Horikoshi, Hiroshi Noguchi
Ibaraki University

A JDMF/Model-1992(JDMF-M92) introduced Object-Oriented paradigm was standardized in 1992. The JDMF-M92 provides only some data construction rules, and for data manipulation it does only some conceptual rules. Therefore we need to prepare the data manipulation rules for a implementation of JDMF-M92.

In this paper ,we implemented the JDMF-M92 using C++ which is the machine independent language and decided some principles for implementation. Further in according to these principles, some methods to generate data, and the data construction rules were implemented.

Keywords : JDMF, Object-Oriented, data modeling facility, C++

1 はじめに

現在様々な分野で利用されているデータベース間のデータ互換性が問題となってきている。また、データベース開発の核となるＤＢＭＳも多数開発されている。そのためにＤＢＭＳの開発のためのデータモデル機能も多数存在している。これらの問題を解決するための対策として、データベースの標準化が考えられてきた。そこで１９９２年に、日本規格協会の情報資源スキーマ調査研究委員会においてオブジェクト指向を採用したＪＤＭＦ－Ｍ９２[1][2]が標準データモデル機能として基準化された。さらに概念スキーマモデル機能の国際標準の１つの候補として提案されている。

基準化されたＪＤＭＦ－Ｍ９２は集合，リスト，属性の入れ子が扱えるなど高い表現能力を持つデータモデル機能であり、特定の実装モデルから独立して記述が行なわれている。またＪＤＭＦ－Ｍ９２はデータ構成規則のみ規定されており、データ操作規則についての特別な規定はされていない。従って、ＪＤＭＦ－Ｍ９２の実現のためにはデータ操作規則を用意する必要がある。

本稿ではＪＤＭＦ－Ｍ９２のデータ操作規則を用意し、基準化されたデータ構成規則とともにＪＤＭＦ－Ｍ９２を機種に依存しないＣ＋＋で実現することを目的としている。

表１，２，３にＪＤＭＦ－Ｍ９２の例を示す。

ここでは、ＪＤＭＦ－Ｍ９２についての詳細は省略する。

2 設計

2．1 開発の目的

ＪＤＭＦに関する実現は今までにSmalltalk/Vなどで行なわれている[3]が、それではSmalltalkだけの閉じた世界となり拡張性に欠けてしまう。従って本稿では、システムに依存しない形でのＪＤＭＦ－Ｍ９２の実現を目的とした。そこで実現に用いる言語を機種に依存しないＣ＋＋とし、ＪＤＭＦ－Ｍ９２の規定外であるデータ操作規則を用意し実現を行う。その際、以下のことを主題として設計を行った。

（１） スキーマ（クラス）の構築
（２） インスタンスの生成
（３） スキーマ（クラス）の消去
（４） インスタンスの消去
（５） オブジェクトの表示
（６） オブジェクトの永続性の確保

2．2 設計方針

本稿において、Ｃ＋＋を用いてＪＤＭＦ－Ｍ９２を実現するにあたり次の設計方針を立てた。

（１）利用形態

簡潔化の為にシングルユーザを仮定する。

（２）扱うデータ型

ＪＤＭＦ－Ｍ９２が想定しているデータとしては文字列，数値，音，ビットマップなどがあるが、データは文字列と整数のみを扱うものとする。

（３）継承

継承が行われるクラスは、名前付オブジェクトクラス，属性付オブジェクトクラスのどちらかのみであると仮定する。

表１：部員

部員				
		連絡先		
部員番号	氏名	住所	電話番号	家族
１００	鈴木一郎	茨城県水戸市･･･	32-1111	{鈴木花子}
２００	山田次郎	茨城県日立市･･･	36-2222	{}
３００	田中太郎	茨城県日立市･･･	36-3333	{田中瞳，田中誠}

表２：属性

属性			
属性ＩＤ	主クラス	属性名	定義域
１０	部員	部員番号	部員番号
２０	部員	氏名	氏名
３０	部員	連絡先	連絡先
４０	部員	家族	家族
５０	連絡先	住所	住所
６０	連絡先	電話番号	電話番号

表３：汎化関係

汎化関係		
汎化関係ＩＤ	スーパクラス	サブクラス
１０	名前付オブジェクト	部員
２０	原子オブジェクト	部員番号
３０	原子オブジェクト	氏名
４０	原子オブジェクト	住所
５０	原子オブジェクト	電話番号
６０	属性付オブジェクト	連絡先
７０	集合オブジェクト	家族

本稿では、「スーパクラスの属性＋サブクラス特有の属性＝サブクラスの属性」として継承を扱うこととする。

（４）クラスの削除

クラスを削除するとき、もしそのクラスが他のクラスのスーパクラスであるか、または属性として使われていたならばそのクラスは削除できないこととする。

（５）階層の高さ

ＪＤＭＦ－Ｍ９２は原子的なオブジェクトをインスタンスとする原子オブジェクトクラスに対して、原子オブジェクトクラスをインスタンスとするクラス「原子オブジェクトクラス」というようにメタクラスを持つ。さらにこのメタクラスに対してのメタクラスを考え、さらにそのメタクラスというように考えていくとメタ階層が無限に生じることになる。従って、ＪＤＭＦ－Ｍ９２を実現する段階でこの階層の高さを決定しなければならない。

本稿では階層の高さを、通常のインスタンスのレベル，そのインスタンスの属するクラス（表のスキーマ）のレベル，このクラスをインスタンスとするクラスのレベルの３階層とした。通常応用データモデルを構築，利用するためには、スキーマを生成するための層，生成されたスキーマが属する層，スキーマの持つ実現値が属する層の３階層を用意すれば十

分と考えられるからである。

（6）C＋＋のクラスとして実現する層

ＪＤＭＦ－Ｍ９２で表（スキーマ）の作成，削除という動作は、いくつかのクラスを生成、消去するということになる。従って実行中に表の作成，削除をするためには、動的にクラスの生成，消去を行う必要がある。しかし、Smalltalkなどと異なりＣ＋＋ではメタ構造が用意されておらず、クラスとはテキスト上にプログラムとして書かれたものだけであるため実行中に新たにクラスを生成，削除することができない。従って（５）で定めた階層を実現するために、Ｃ＋＋のクラスとして何を記述するかを決定しなくてはならない。

そこでＣ＋＋のクラスとして（５）の最上層をクラスとし、それ以外をインスタンスもしくはそれに準ずるものとした。

（7）各階層の実現方法

先に述べた３階層をそれぞれどのように表すかを決定する。まず（６）で述べたように最上層はクラスとしてプログラム上に記述する。次に中位層は最上層で定義されたクラスのインスタンスで実現するとする。最下層のオブジェクトは、中位層に生成されたインスタンスのインスタンスメソッドによりメモリ上に領域を取りそこに記憶するものとする。このデータはリスト構造を持ち、中位層に生成された「名前付オブジェクトクラス」のインスタンスがそのリスト構造を管理する。以上の設計により、Ｃ＋＋では扱えない「メタクラス，クラス，インスタンス」という階層を「クラス，インスタンス，メモリ上の純粋なデータ」という形で実現することが可能となる。その様子を図１に示す。

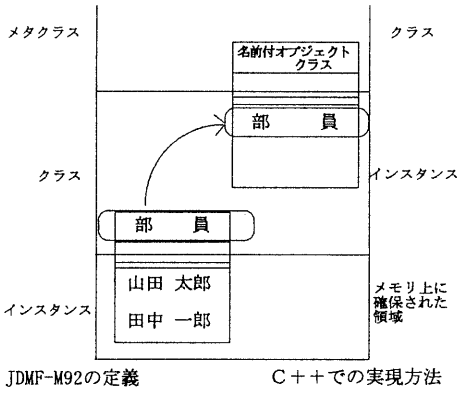

図１ 実現する３階層

上記の構造を用いる事で、ＪＤＭＦ－Ｍ９２の操作を次のように実現することが可能となる。

・スキーマの構築におけるサブクラスの生成

スキーマを構築するには、まず名前付オブジェクトクラスのサブクラスとしてメタクラスであるクラス「名前付オブジェクトクラス」のインスタンスを生成する。次にとその属性を構成するクラスを対応するクラス（例えば、クラス「原子オブジェクト」）のサブクラスとして、クラス「原子オブジェクトクラス」などのインスタンスを生成する。そして、クラス「汎化関係」とクラス「属性」にその情報を持つインスタンスを生成することにより行なう。

・表への実現値の挿入

表に対して実現値を生成したい場合には、その表を表している名前付オブジェクトクラスの属性となっているインスタンスをクラス「属性」から検索し、そのインスタンスの持つメソッドによりメモリ上

に領域を確保しそこに記憶する。確保された領域はリスト構造を構成し、その先頭のアドレスを名前付オブジェクトクラスが持つ。

（8）データ構造

最下層に生成されるデータを記憶するためのデータ構造としてリスト構造を用いる。リスト構造は、クラス「属性」用，クラス「汎化関係」用，宣言オブジェクトクラス用の３種類を用意する。宣言オブジェクトクラス用のリスト構造を図２に示す。

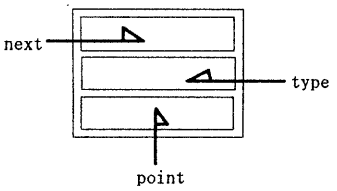

図２ リスト構造

データ構造は次の３つのメンバを持つ。

・next

次のノードを指すポインタ

・point

構造上どのようなオブジェクトがつながるかわからないので、汎用ポインタ（void*）を用意する。

・type

pointにつながるオブジェクトの種類を示す。

type	オブジェクトの種類
0	ユーザの定義した名前付オブジェクトクラス
1	原子オブジェクトクラスのインスタンス（文字列）
2	原子オブジェクトクラスのインスタンス（整数）
3	集合オブジェクトクラスのインスタンス
4	リストオブジェクトクラスのインスタンス
5	属性付オブジェクトクラスのインスタンス
6	名前付オブジェクトクラスのインスタンス

２．３ 設計結果の例

今まで述べた設計に基づいた構造で、表１，２，３の例を表した結果を図３，４，５に示す。

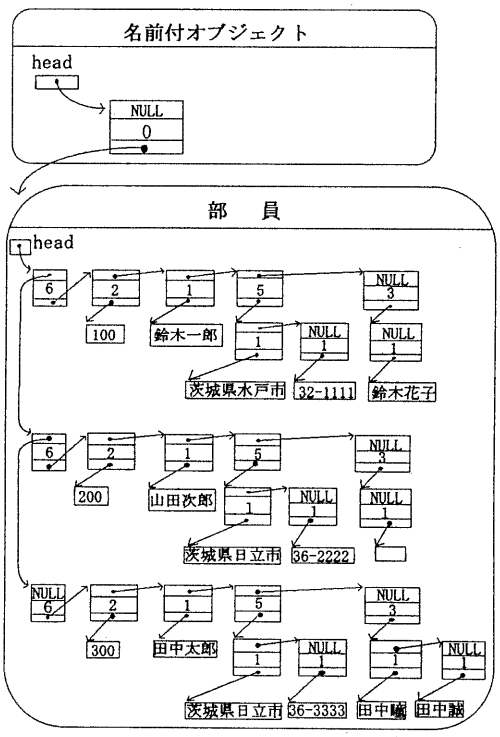

図３ 名前付オブジェクトクラスの例

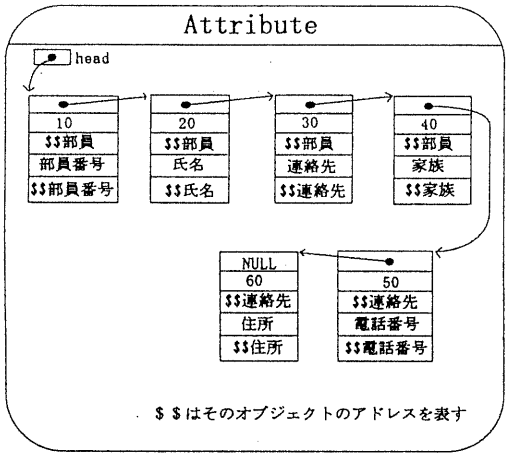

図４ 属性の例

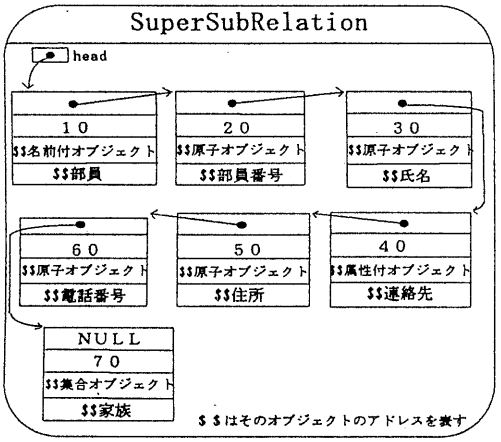

図５ 汎化関係の例

３ ユーザインタフェース

今回の実現において、ユーザのスキーマへの操作はメニュー形式で行う。

３．１ CREATE

CREATEはユーザがスキーマを定義するためのメニューである。このメニューが選択されたときは、クラス「名前付オブジェクト」の持つメソッドを呼び出し、スキーマの構築を行う。

そのメソッドでの処理を以下に示す。

１）名前付オブジェクトクラスの生成

・新しく生成

ユーザがクラス名の入力を行い、名前付オブジェクトクラスの生成が行なわれる。

・既にあるクラスから継承

ユーザはスーパクラスのクラス名の入力を行う。次に新たに生成する名前付オブジェクトクラスのクラス名を入力し、生成が行なわれる。

２）属性となるクラスの決定

・新しく生成

ユーザは属性となるクラスの種類（集合オブジェクト，原子オブジェクト，・・・）を決定する。次にクラス名，属性名の入力を行う。

・既にあるクラスの利用

ユーザは属性となるクラスの種類（集合オブジェクト，原子オブジェクト，・・・）を決定する。次に属性名の入力を行う。

３．２ INSERT

INSERTは構築されたスキーマの実現値を生成するためのメニューである。ユーザがこのメニューを選択したときの処理は以下の通りである。

１）構築されている名前付オブジェクトクラスを表示

２）ユーザはこの中から選択を行い、実現値を生成したいクラスのクラス名を入力する。

３）選択されたクラスの属性となっているクラスをクラス「属性」から検索しそのインスタンスを生成し、リスト構造を構成する。

３．３ SELECT

SELECTはスキーマの持つ実現値を表示

するためのメニューである。SELECTが選択されたときの処理を以下に示す。

1）構築されている名前付オブジェクトクラスの表示を行う。
2）ユーザはクラス名を入力し、選択を行う。
3）入力されたクラスのリスト構造に従い、その実現値の表示を行う。

3．4 DELETE

DELETEはスキーマの持つ実現値を削除するためのメニューである。このメニューが選択されるときの処理を以下に示す。

1）構築されている名前付オブジェクトクラスの表示を行う。
2）ユーザはクラス名を入力し、選択を行う。
3）次に削除したいオブジェクトの行を入力する。
4）そのオブジェクトを削除する。

3．5 DROP

DROPはユーザが定義したスキーマの削除，クラスの削除を行なうためのメニューである。このメニューが選択されるときの処理を以下に示す。

1）ユーザにより生成されたクラスを表示する。
2）ユーザは表示されているクラスの中から削除したいクラスのクラス名を入力する。
3）選択されたクラスが他のクラスのスーパクラスでなく、他のスキーマで属性として使用されてなければ削除する。

3．6 QUIT

メニューが選択されると、クラス「名前付オブジェクト」，クラス「属性」，クラス「汎化関係」のセーブ用のメソッドがそれぞれ実行され、終了する。
この保存された情報は次回の開始時にロードされ、オブジェクトの永続性が確保される。

4 まとめ

本研究ではメタ階層の高さを３層とし、最上層をクラス，中位層をインスタンス，最下層をそのインスタンスのインスタンスメソッドにより生成されるデータとするなどの実現のための方針を決定した。そして、この方針と規定されたデータ構成規則に従うデータを生成するためのメソッドを用意し、データ構成規則とともにＣ＋＋によるＪＤＭＦ－９２の実現を行なった。さらに、メニュー形式のユーザインタフェースを使い、実現したＪＤＭＦ－Ｍ９２による応用データモデルの構築，利用を可能とした。またオブジェクトの永続性を考え、セーブ，ロード機能を実現した。

また今後の課題として以下に述べる様なことが挙げられる。

1）扱える原子オブジェクトの種類を増やす。今回の実現で扱える原子オブジェクトは文字列と整数に限定したが、今後この種類を増やすことが考えられる。
2）本稿におけるセーブ，ロード機能はそれぞれ終了時，起動時に自動的に働き、ユーザは関知しない。このセーブ，ロードを行なうメソッドを拡張し、commit，abortを実現することが可能であると思われる。

謝辞

本研究を行なうにあたり、多くの有意義な御意見を下さった茨城大学の加納幹雄教授，上田賀一講師に深く感謝します。

参考文献

[１]日本規格協会，データモデル機能 JDMF/Model-1992，１９９３年

[２]佐藤英人，JDMF/M-92 オブジェクト構成，１９９３年：第２回データエンジニアリングフォーラム

[３]野口宏，尾関美明，穂鷹良介，ＪＤＭＦの実現，１９９２年：情報処理学会研究報告92-DBS-87

[４]R.G.G.Cattell， The Object Database Standard ：ODMG-93，１９９４年：Morgan Kaufmann Publishers