マルチメディア通信と分散処理 42－12
（１９８９．５．１９）

リレーショナル型データベースを用いた OSIディレクトリ情報ベース(DIB)の実現と評価

西山 智　小花 貞夫
KDD 上福岡研究所

本稿では汎用のリレーショナル型データベース管理システム(RDB)を用いたOSIディレクトリ情報ベース(DIB)の実現方法とその評価について報告する。実現するDIBに木の深さ等の制限を設けず、かつ検索効率を向上させるため、非正規形のRDBスキーマをデータ構造として使用した。またDIBの構造やRDBのスキーマ自身をデータとして扱い、プログラム構造から分離することでDIBの構造の変更に柔軟に対処できる。評価の結果、大規模なDIBにおいても実用的な検索速度が得られた。

Implementation of OSI Directory Information Base (DIB) using Relational Database Management System

Satoshi NISHIYAMA　　*Sadao OBANA*
KDD Kamifukuoka R & D Labs.　2-1-15, Ohara, Kamifukuoka, Saitama, 356

In this paper, an implementation methodology for the OSI Directory Information Base (DIB) using a relational database management system (RDB) and its evaluation are discussed. In this methodology, the DIB structure are realized without any limitations. In order to improve the access performance, non-3NF (3rd Normal Form) schema are used in the RDB. Furthermore, the data structure of the DIB (Directory schema) and the RDB schema are separated from the program structure and are defined as data for the program. Therefore, a change of the Directory schema only needs the redefinitions of these data. Finally, the result of the evaluation shows the practicability of this methodology.

1. はじめに

近年、FTAM(ファイル転送、アクセス及び管理)やMHS(メッセージ通信処理システム)といったOSI通信システムが実用化されつつある。OSI通信システムを網構成の変更に柔軟に対処可能とするためには、応用エンティティ名称からプレゼンテーションアドレスを得たり、MHSでO/R名からO/Rアドレスを得るといったディレクトリ機能を提供するデータベース(ディレクトリシステム)が必要となる。OSIディレクトリシステムは、OSI通信システムのみならず、電話やISDNといった様々な通信サービスに関する情報を提供可能な、汎用のディレクトリシステムである。

筆者らはこれまでに、OSI通信システムのソフトウェア群(MHS、FTAM等)の実装を進めてきた。これらのソフトウェア群及び一般の通信利用者が必要とするディレクトリ機能を提供するために、筆者らはVAX/VMS上にOSIディレクトリシステムの実装を行った。[1][2][3][5]この実装では、OSIディレクトリシステムのディレクトリ情報ベース(DIB)を汎用のリレーショナル型データベース管理システム(RDB)を用いて実現している。DIBは、1)全体のデータ構造が木構造をなすこと、2)多種類のデータを扱い、かつデータの種類は変更がありえること、といった特徴を持つ。筆者らはこれらの特徴を考慮し、非正規形のデータ構造を使用し検索速度を向上させるとともに、データ構造をプログラムから独立させ、データ構造の変更に容易に対処可能なソフトウェア構成とした。OSIの他の応用(例えばOSI管理)においても、DIBのような木構造のデータ構造を持つものが増えてきており、RDBを用いた木構造データの実装手法は今後重要となる。

本報告では、このDIBの実現手法、及びその評価について報告する。

2. DIBの構造

2.1 DIBの構造と要素

DIBは、図1に示すように、各オブジェクトの名前管理(明確に識別できる名前を与えること)のために、また容易に分散管理できるようにするために、ディレクトリ情報木(DIT)と呼ばれる木構造で表現される。DITにおける木の節(ふし)はエントリを、また木の枝はエントリ間の従属関係を表す。

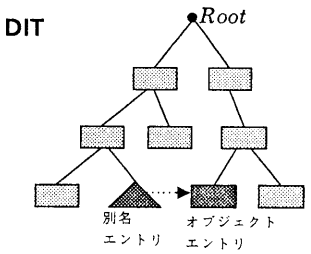


図1　DIT(ディレクトリ情報木)

エントリにはオブジェクトエントリと別名エントリがある。オブジェクトエントリはひとつのオブジェクトを表現し、それに関する情報を持つ。またオブジェクトによっては、オブジェクトエントリの他にオブジェクトに別名をあたえる別名エントリも存在する。

各エントリは、図2に示すように属性の集合から構成されている。属性は例えば、人名、住所、電話番号などオブジェクトに関する各種の情報を表し、属性情報の種類を示す属性タイプとその属性の実現値である属性値から構成される。

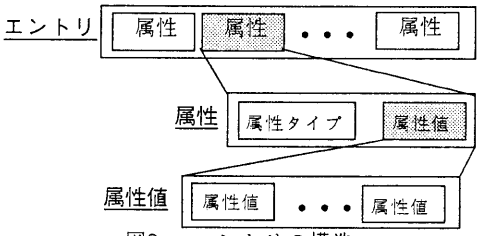
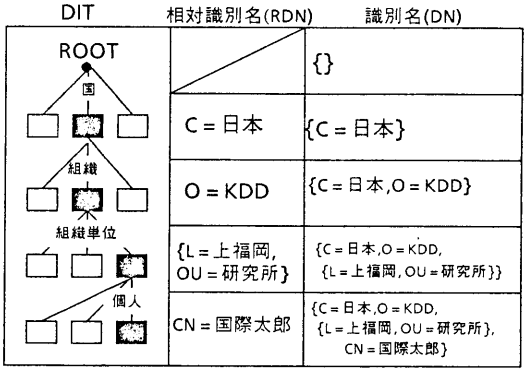


図2　エントリの構造

ディレクトリでは、全てのエントリは識別名(DN)により一意に識別され、検索などのアクセスを行う時にこれを指定する。エントリの識別名は、図3に示すようにエントリの直接上位のエントリの識別名とそれに対する相対識別名(RDN)で表される。あるエントリの直接下位の全てのエントリのRDNは互いに異る必要がある。



C : 国名　　L : 地域名　　CN : 通称
O : 組織名　　OU : 組織単位名

図3　識別名の構造

2.2 ディレクトリスキーマ

DITは、任意に構成できるものではなく、ディレクトリスキーマと呼ばれる以下の4つのDIBの構造上の制約の規定に従って構成される。

(1) DIT構造　DIT木構造でとりえるオブジェクトクラス間の上下関係および各オブジェクトクラスでRDNとして使用可能な属性タイプを定義する。

(2) オブジェクトクラス　各オブジェクトクラスで必須な属性タイプとオプショナルな属性タイプを定める。

(3) 属性タイプ　各属性タイプの属性シンタックスおよび属性値が複数回出現可能かを定義する。
(4) 属性シンタックス　各属性の値の型および比較の規則を定義する。

これら4つのディレクトリスキーマとDITとの関係を図4に示す。これらのディレクトリスキーマは勧告/標準で規定されている。しかし、勧告/標準で規定されているディレクトリスキーマはさまざまな応用に共通的なもののみであり、応用固有のディレクトリスキーマは応用毎に追加定義されることとなっている。例えばMHSでは、MHSに固有のオブジェクトクラスMTA(メッセージ転送エージェント)や属性タイプO/Rアドレス等を定義している。

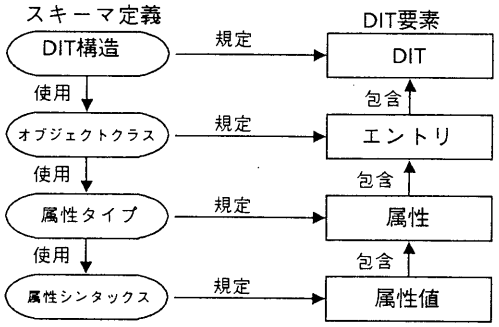


図4　ディレクトリスキーマとDIT構造の関係

2.3 分散処理知識

ディレクトリシステムエージェント(DSA : DIBを分散して保持し、ディレクトリサービスを協調して提供する)の分散処理のための、DIBのどの部分がどのDSAに収容されているかに関する知識(Knowledge Reference)もDITと同様に木構造(知識木:KT)で表現される。この知識は5種類ある。

① 内部リファレンス(Internal reference)
自DSA内に存在するエントリに関する知識(内部への論理的なポインタ)

② 下位リファレンス(Subordinate reference)
自DSA内に存在するエントリの下位エントリが他のDSAに存在する場合、そのエントリに関する知識(そのRDN、DSAの名前とPSAPアドレス)

③ 上位リファレンス(Superior reference)
他の知識が全く使用できない場合の最終的な処理依頼先DSAの知識(DSAの名前とPSAPアドレス)

④ 不特定下位リファレンス(Non-specific subordinate reference)
②の場合にその下位の名前が不明で存在のみが分かっている場合の存在先DSAに関する知識(DSA名およびDSAのPSAPアドレス)

⑤ クロスリファレンス(Cross reference)
(アクセス回数が多い等の運用上の理由により)DITの自DSAの持つ部分木とは直接関連のないエントリに関する知識(そのDN、DSA名およびDSAのPSAPアドレス)

従って、DIBを実現する場合には、同時にこれらのKTも実現する必要がある。

3. RDBスキーマの設計

RDBを用いてDIBを実現する場合、DIBのデータ構造およびDIBに対する操作の特徴を考慮して、効率のよいRDBスキーマを設計することが肝要である。2節より、DIBのデータ構造の特徴として以下の2点が挙げられる。

a) エントリを節とした木構造で表現され、木の深さには制限がない。また、エントリ内の各属性に対する属性値は複数の値を持てる。
b) DIB構造はディレクトリスキーマにより規定される。ディレクトリスキーマはオブジェクトクラスの追加等により変更されうる。

また、DIBに対する操作の特徴としては、

c) 検索回数に比較して更新回数は少ない。

以下にこれらの特徴をふまえたRDBスキーマの設計について述べる。

3.1 設計の基本方針

DIBを実現するにあたっては以下のような方針を設けた。

1) DIB(DIT)構造に可能な限り制限を設けない。DITの木の深さやエントリの属性値の数に制限を設けない。また勧告/標準で定義されている全てのオブジェクトクラス、属性タイプを扱う。
2) 格納方法は格納効率、更新速度より検索速度を重視する。通信システムで接続の際に使用されることや一般の端末利用者がアクセスすることを考慮すると、検索速度の向上は重要である。

3.2 KT及びDITのスキーマ設計

一般に木構造では、ある節とその下位の節は1:nの関係となり、キーによるテーブル間の結合(join)関係により表現できる。この場合、木構造をm段検索するためにはm回のテーブル間の結合操作が必要となる。一方、木の枝を複数段まとめて検索キーとすると、RDBでの正規化条件を満たしていない非正規形のデータ構造となり、冗長なデータを持つためデータの格納効率が悪くなる。しかし、木構造を複数段検索するのに必要なデータベース操作の数は少ない。これらの表現方法の比較を表1に示す。DIBに対する操作の特徴として、更新操作の頻度は検索操作の頻度に比較して少ないことを考慮すると、後者の表現方法が望ましいと考えられる。

DIT木構造の表現法	テーブル構造	検索速度	データ格納効率
1段毎にテーブルのタプルとして表現	正規形	×	○
多段まとめてテーブルのタプルとして表現	非正規形	◎	×

表1　DIT木構造表現方法の比較

この方針から、図5に示すDIT/KT構造を使用することとした。DITの木構造はKTの内部リファレンス、下位リファレンスとともに、REFテーブルにより表現される。REFテーブルは、

- テーブルに複数(現在5)段のRDNをカラムとして持たせ、一度に複数段の上位/下位関係を検索可能とし、検索速度を向上させる。
- RDNは簡易な構文の文字列に変換されてカラムに格納される。RDNはASN.1のSET型で定義されておりその符号化した値は一意ではない。ここではRDBの検索機能を使用するために、簡易な構文化により値が一意に定まるよう正規化している。
- DITでのエントリ間の上下関係を表現するために上位エントリのREF番号(KTで各エントリに付与したユニークな識別子)を結合キー(DIT上位REF番号)として持たせた。これはList操作やSearch操作でDIT木構造を検索する場合に使用される。
- 1度に複数段の枝が検索されるため、複数段毎の木の上下関係を表現するための複数段上位のREF番号も結合キー(上位REF番号)として持たせた。

また、KTを構成する他の知識は、REFテーブルの他、SUPRテーブル(不特定下位リファレンスを格納)、NCPテーブル(内部リファレンスの一部及びクロスリファレンスを格納)及びNSSRテーブル(不特定下位リファレンスを格納)により表現される。

3.3エントリの設計

1)エントリの構造

エントリは検索速度を重視し、図5に示すような構造をとる。

- オブジェクトクラス毎にテーブルを対応させる。
- エントリに対する検索は、1回のデータベース操作で実現可能とするのが効率的である。このためユニバーサルリレーション的にエントリの取りえる全ての属性をカラムとして設定する。
- エントリには一般に任意の数の属性値が含まれるが、一般的なエントリはそう多くの属性値を持たない。このため、標準的なエントリが含む値の最大数をオブジェクトクラスとその属性毎に設定し、その数のカラムを用意する。
- 属性毎にあふれ属性値テーブルを設けて、エントリの属性値の数が最大数をこえた場合に、そこに格納する。各オブジェクトクラスに対応するテーブルには、値のためのカラムに加えて、値があふれたことを示す拡張フラグを設ける。

このような構造をとった場合、各オブジェクトクラスに対応するテーブルも非正規形のテーブルとなる。これにより更新操作では、データ一貫性を保つための操作が必要となる。例えば、あふれ値が存在する属性の値の1つを削除した場合、値をつめる必要がある。しかし、一般的な属性値数を持つエントリに対する検索は1回のデータベース操作で済み、全体的な検索速度が向上する。

2)属性値の格納

- 各属性値に対するカラム幅は基本的に勧告/標準で規定されるものを使用する。
- 各属性値をテーブルのカラムに格納する方法は、以下の種類がある。
  1) 値がASN.1の(文字列)プリミティブの場合、文字列として格納する。CaseIgnore文字列の場合、大文字に変換し格納することで一意性を保った。
  2) 値がASN.1の構文で表現されており、属性値に対して値の比較方法が定義されている場合、①データの格納効率がよいこと、②表

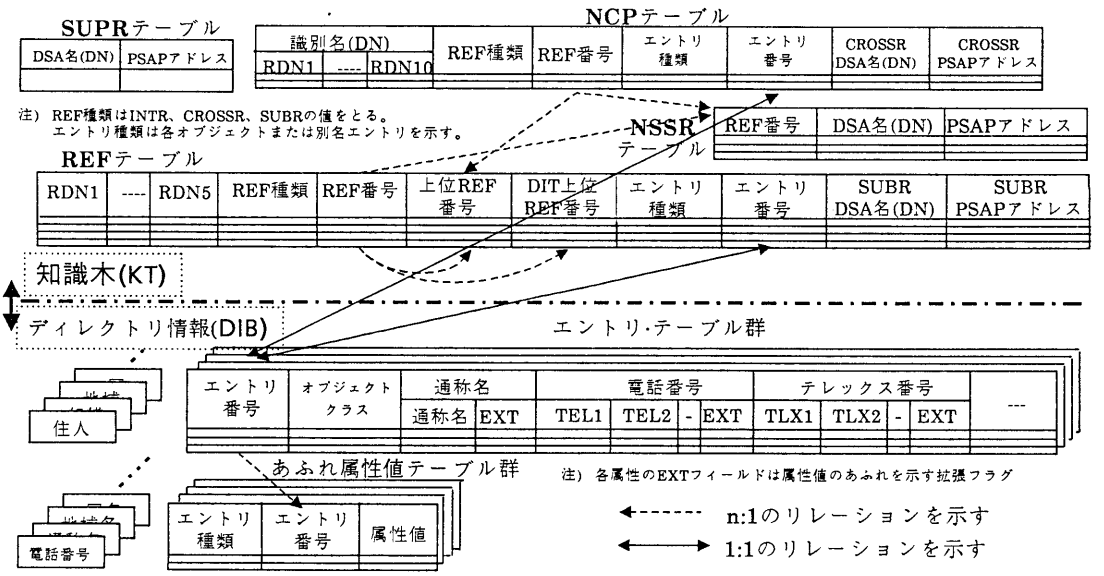


図5 DSAにおけるディレクトリ情報のテーブルとリレーション

現が一意であること、③データベースを直接操作する場合にも判読が容易であること等を考慮し、簡易な構文に変換し文字列として格納する。簡易構文の例を図6に示す。

3) 値がASN.1の構文で表現されており、属性値に対して値の比較方法が定義されていない場合、ASN.1の符号化規則に従って符号化したものをそのまま格納する。

4) 電話番号(TelephoneNumber)属性では、比較の際に空白と'-'を無視することになっている。このため値を一意に正規化するには格納時にこれらの文字を取り除く必要がある。しかし、この正規化を行うと値を読み出した時に利用者の判読性が低下するため、正規化を行っていない。比較はRDBの機能によらず、プログラムにより実現する。

識別名の構造(ASN.1)

```
DistinguishedName ::=          SEQUENCE OF RDN
RDN ::=    SET OF AttributeValueAssertion
AttributeValueAssertion ::=    SEQUENCE OF { AttrType,
                               AttrValue }
```

簡易構文表現例

```
(6:"JP", 10:"KDD",
    (7:"KAMIFUKUOKA",11:"R & D LABS."),
    11:"COMPUTER-TSUUSHIN LAB.",
    3:"TARO KOKUSAI")
```

図6 簡易構文の例(識別名)

4. ディレクトリ操作の実現

ディレクトリ操作は大きく1)名前解読、2)操作実行の2段階で処理される。各段階では、3節で決定したデータ構造をもとに以下のようにディレクトリ操作が実現される。

4.1 名前解読

名前解読段階ではKTの4テーブル(NCP、REF、NSSR、SUPR)を用いて操作対象エントリが自DSAに格納されているかどうかを判定する。判定の概略は以下の①~⑦のステップで行う。

① 識別名をキーにして、NCPテーブルから操作対象エントリが含まれるDIT部分木を検索する。みつからない場合、SUPRテーブルの値によりチェイン(他のDSAに操作を転送)する。

② 該当した部分木が他DSAにある場合、NCPテーブルのクロスリファレンスによりチェインする。

③ 該当した部分木が自DSAにある場合、識別名のうち未検索の相対識別名(RDN)をキーにREFテーブルを用いて操作対象エントリを検索する。REFテーブルによるエントリの検索はRDNを最大5段まとめてキーとして検索を行う。5段以上RDNがある場合は複数回にわけて検索を行う。

④ 検索結果のエントリのリファレンス種類が下位リファレンスの場合、チェインを行う。

⑤ 検索結果のエントリのリファレンス種類が内部リファレンスの場合、エントリ種類·番号をもとに操作実行段階に移る。

⑥ ③で検索が失敗した場合NSSRテーブルを検索し、不特定下位リファレンスが存在した場合それらをもとにマルチキャスト(他の複数のDSAに同じ操作を転送)する。

⑦ ⑥で不特定下位リファレンスが存在しない場合、その識別名を持つエントリは存在しないためエラーとする。

4.2 操作実行

操作実行段階では名前解読段階で得られた対象エントリのエントリ種類とエントリ番号(キー)をもとにエントリテーブルとそのタプルを決定し、値の読み出し·比較·更新等を行う。

1) Read操作、Compare操作

対象タプルの値の読み出し/比較を行う。操作対象の属性にあふれ値がある場合は、あふれ属性値テーブルの読み出し/比較も行う。

2) list操作

REFテーブルのみを用いてエントリ間の上下関係を調べ、対象下位エントリのRDNを得る。

3) Search操作

Read操作と同様に対象タプルの値の読み出しを行う。操作対象の属性にあふれ値がある場合は、あふれ属性値テーブルの読み出しも行う。また、複数エントリに対する操作を指定された場合、List操作と同様REFテーブルを用いてエントリ間の上下関係を得て処理を行う。

4) 更新操作(AddEntry操作、RemoveEntry操作、ModifyEntry操作、ModifyRDN操作)

エントリテーブル及びあふれ属性値テーブルの更新を行う。ModifyEntry操作では、属性値の削除に伴い、あふれ属性値テーブルからエントリテーブルへの値の詰め替えも必要となる場合がある。また、他の3つの操作によりエントリの追加/削除/名前変更が行われた場合、KTを構成するREFテーブルの内部リファレンスも更新する必要がある。

4.3 スキーマのプログラム独立化

3.1節に述べたように、オブジェクトクラスの追加定義やDIT構造定義の変更によりディレクトリスキーマは変更される可能性がある。また、KT及びDIBを実現する非正規形のRDBスキーマは、検索効率等の評価により変更されることがある。さらに、属性値をテーブルに格納する際のローカルなシンタックスやカラムの長さといった実装上の情報についても変更可能であることが望ましい。従ってこれらの変更がプログラムに影響を及ぼさないように、ディレクトリスキーマ及びRDBの非正規形データ構造に関する情報をデータとして定義しプログラム構造から独立させることとした。

この方針に基づき、これらの情報はRDB上にデータとして表現され、プログラムはその情報を検索してディレクトリ操作を実行する。勧告のスキーマ定義に沿ってデータを①DIT構造定義、②オブジェクトクラス定義、③属性タイプ定義、④

属性シンタックス定義に分類し、さらに、本実装特有の情報として、⑤エントリテーブルに収容する各属性に対して用意するカラム数、⑥エントリテーブルへ格納される属性値のローカルなシンタックスおよびカラム長を追加した。

これら6種類の情報はRDB上のスキーマテーブルとして、それぞれDIT構造定義テーブル(①)、オブジェクトクラス定義テーブル(②,⑤)、属性タイプ/シンタックス定義テーブル(③,④,⑥)に収容される。

オブジェクトクラス毎のエントリテーブルおよびあふれ属性値テーブルは6節で述べるツールにより、スキーマテーブルの情報(各オブジェクトクラスの持つ属性、各属性値のカラム数およびカラムの長さ、オブジェクトクラス/属性/カラム数から合成される各属性値のカラム名)から生成される。生成されたテーブルやそのカラムの名前は、オブジェクトクラスや属性に割当てられた識別番号から決定できるようになっている。従って、これらのディレクトリスキーマ情報と上記のテーブル/カラム名の生成規則により、ディレクトリ操作に必要なデータベース操作が生成できる。図7にスキーマのテーブルおよびエントリテーブルとの関係を示す。

5. 実装

DIBの実装はVAX/VMS上の汎用RDBパッケージORACLEを用いて行った。3節の設計に基づきDIB、KT及びディレクトリスキーマを構成するテーブル群はRDBのテーブルとして実現した。また、検索速度向上のため、システム運用中に変更されない情報(KTのNCPテーブル、SUPテーブル及びディレクトリスキーマ)については、システム立上げ時に主記憶にコピーされ、データベースアクセスなしに参照できるようにした。

ディレクトリ操作は、DIBプロセス[1]で実行される。DIBプロセスは、操作の複数同時実行を実現するために、システムに複数用意されており、このため前述のディレクトリスキーマ等の主記憶上のコピーはこれらプロセス間で共有される。また、これらのRDB上のテーブルを保守するためのツールが用意されている。図8にRDB上のテーブル、共有メモリ、DIBプロセス及び管理ツールの関係を示す。

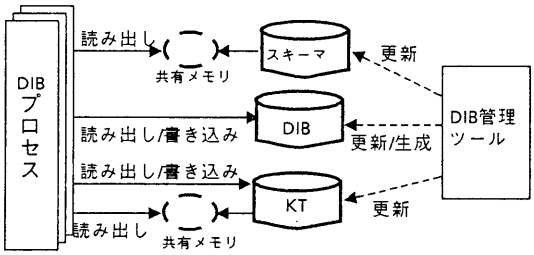


図8　テーブル、DIBプロセス及びDIB管理ツールの関係

6. DIB管理ツール

DIBは一種のデータベースと考えられるため、データベースへのローディング機能やバックアップ機能といった保守運用機能が必要となる。使用したRDB(ORACLE)は、テーブル単位でのローディング/バックアップ機能を持っている。しかし、オブジェクトクラス及び属性タイプ別にテーブルを設けたことにより対象となるテーブル数が多い(約60)こと、またディレクトリスキーマの変更によりRDBのスキーマも変更されることの理由により、ローディング/バックアップ機能をツールとして設けた。さらに、KTの管理は勧告/標準では実装にまかされているため、KTについてもツールにより管理を行うこととした。本実装においては、次の3つのツールを用意した。

(1) DITMNG

ディレクトリスキーマのローディング/バックアップを行う。またそのスキーマを参照して、

ディレクトリスキーマを表すテーブル

DIT構造定義テーブル

オブジェクトクラス	上位オブジェクトクラス	識別名として必須な属性	識別名として使用してもよい属性
Organization	Root-Country-Locality	OrganizationName	

オブジェクトクラス定義テーブル

オブジェクトクラス	サブクラス	属性(1)			-	属性(2)			-	属性(3)		
		タイプ	必須	カラム数	-	タイプ	必須	カラム数	-	タイプ	必須	カラム数
Organization		LocalityName	MAY	1	-	OrganizationName	MUST	1	-	TelephoneNumber	MAY	3

属性タイプ/シンタックス定義テーブル

属性タイプ	シンタックス	複数の値を持つか	制限長	マッチングルール	ローカルシンタックス	ローカル制限長
OrganizationName	CaseIgnoreString	YES	64	Equality/Substring	Upper Case Character	64

組織のエントリテーブル

エントリ番号	地域名		---	組織名		---	電話番号				---
	地域名(128)	EXT		組織名(64)	EXT		TEL1(32)	TEL2(32)	TEL3(32)	EXT	

注)各属性の()内の数字はカラム長、EXTはあふれを示すフラグをしめす。

図7　ディレクトリスキーマとエントリテーブルの関係

ディレクトリシステムが必要とするテーブル群の生成/削除を行う。

(2) KTMNG

KTを構成する5種類の知識のローディング/バックアップを行う。

(3) DIBMNG

DIBを木構造ごと、ローディング/バックアップを行う。ローディングの際はKTの一部(内部リファレンス)も併せて生成される。

これらのツールが使用する外部ファイルは、保守性を考慮しテキストファイルとし、通常のエディタにより作成や編集が行えるようにした。図9に外部ファイルのフォーマット例としてディレクトリスキーマファイルの例を示す。

```
/* DIT STRUCTURE DEFINITION */
OBJECT = 4:Organization,
    SUPERIOR = (0:Root, 2:Country, 3:Locality),
    MANDORY = (10:OrganizationName);
/* OBJECT CLASS DEFINITION */
OBJECT = 4:Organization,
        SUPERCLASS = 0,
        ATTRIBUTE = (0:ObjectClass, MUST, 1),
        ATTRIBUTE = (7:LocalityName, MAY, 1),
        ATTRIBUTE = (9:StateOrProvinceName, MAY, 1),
        ATTRIBUTE = (10:OrganizationName,MUST,1),
                .
        ATTRIBUTE = (20:TelephoneNumber,MAY,3),
                .
        ATTRIBUTE = (35:UserPassword,MAY,1);
/* ATTRIBUTE TYPE & SYNTAX DEFINITION */
ATTRIBUTE = 10:OrganizationName,
            SYNTAX = 4,MULTI = YES,LENG = 64,
            MATCH = (EQUALITY,SUBSTRING),
            LOCALSYNTAX = 2,LOCALLENG = 64;
```

図9　外部ファイルフォーマット例(ディレクトリスキーマ)

7. 評価および考察

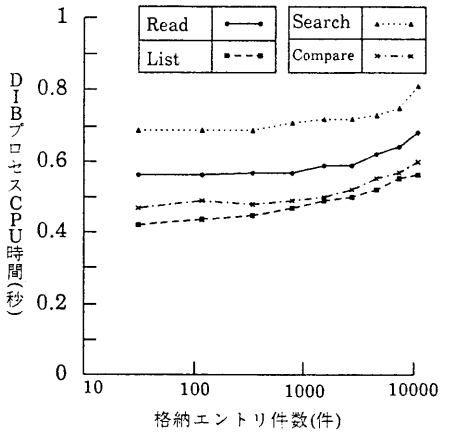
7.1 検索速度について

DIBの検索効率に関する評価は、VAX8700上で行った。

1) KTのテーブル構造と名前解読処理速度

図10にDIBに格納されるエントリ件数と検索速度のグラフを示す。図中の値は単一エントリに対する各操作をDIBプロセスが処理するのに必要なCPU時間を示している。利用者の応答時間はDSA直結のDUAの場合、このCPU時間にDSAプロセスで処理に要するCPU時間(単一エントリに対する操作で約0.06~0.08秒)を足した値と殆ど等しい。

図11は、操作対象エントリの識別名を構成するRDNの段数と検索速度を示している。名前解読処理でDITを5段まとめて検索しているため、検索時間も5段毎に変化している。個々のRDNはDIB全体としてもかなりユニークである。このため、RDBのインデックスを用いた効率良い検索が行え、複数段まとめたことによる検索条件の複雑さは検索速度に余り影響していない。なお、この例でRDNが1段の場合は、部分木の頂点エントリとして知識がNCPテーブルに格納されており、名前解読処理が共用メモリ上のコピーに対するアクセスで済むため、他の場合に比較して検索に要する時間が小さい。図10の結果より、エントリ件数の増加に対して名前解読速度の低下は少ないことが分かる。



測定条件(単一エントリに対する平均値)

Read操作	RDN5段のエントリ、全属性読み出し
List操作	RDN4段のエントリ
Search操作	RDN5段のエントリ、単一エントリ操作、フィルタなし、全属性読み出し
Compare操作	RDN5段のエントリ、電話番号の比較

図10　DIBドライバの処理時間

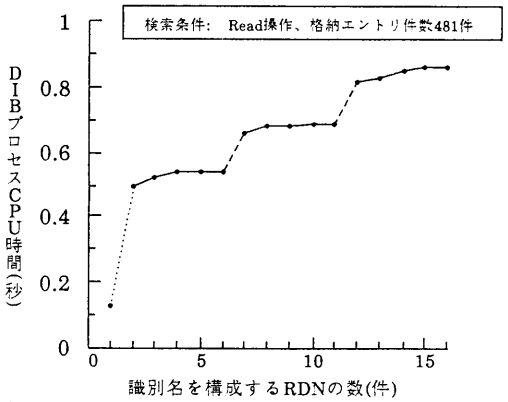


図11　識別名を構成するRDNの数と名前解読処理速度

また、図11の結果より、複数段まとめて検索することにより、木構造の検索が効率的に行えていることが分かる。これらのことから、本手法による非正規形データ構造の検索の効率性が実証できた。

2) エントリのデータ構造

各検索操作は通常の場合、エントリに対する1回のデータベース操作で全ての属性値が読み出せるが、値の数が多い場合は、あふれ属性値テーブルも検索する必要がある。このあふれ属性値テーブルの検索に要する時間はエントリ件数10000の場合、1テーブル当り約0.03~0.05秒である。この値を参考にすると、エントリを表現するのに正規形表現をとった場合、例えばRead操作で全属性読み出しを指定した場合、各オブジェクトクラスは20程度の属性を含みうるため、処理速度は本実装に比較して約20(読み出し属性数)×約0.03~0.05秒遅くなることが予想される。このことから、本実装で用いたデータ構造の有用性が実証できる。

3)複数エントリに対するSearch操作

Search操作では、REFテーブルを用いてDIT上の上位/下位関係を得て、複数エントリに対する操作を実行する。図12に検索の結果のエントリ件数と処理時間の関係を示す。格納エントリ件数にかかわらず、結果エントリ1件当たり0.5秒の処理時間を必要としている。Read操作やSearch操作で単一エントリに対して名前解読を除く操作実行処理に0.3~0.4秒程度必要としていることから、この処理時間はDIT構造の検索ではなく、値の読み出しと加工にかかっていると考えられる。従って検索結果の件数が増加すると、この程度の処理時間は必要であると考えられる。

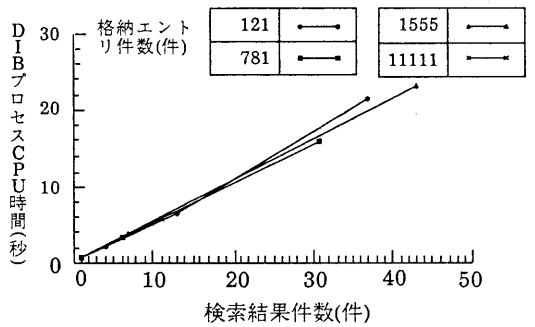


図12 Search操作処理時間(複数エントリ検索時)

7.2格納効率について

本実装のような非正規形のRDBデータ構造を使用する場合、冗長なデータがテーブル中に存在するため、正規形のデータ構造に対してより多くの格納領域を必要となる。

1)KT及びDIT

DITをX段まとめて検索可能とすると、各内部リファレンスは最大X個のRDNを持つ必要がある。したがって、1段毎に検索する方法(正規形データ構造)に比べて、RDN長が無限大の場合、格納効率は最悪1/Xとなる。しかしながら、RDNは一般的にそう長くはないこと等の理由により、評価に用いたテストデータ(X=5、エントリ数約11000件)で格納効率は約1/2であった。

2)各エントリの格納効率

各エントリの格納効率については、本実装で使用したORACLEではNULL値が格納領域を使用しないため、本実装の方が正規形のRDBスキーマをとる場合に比較して各属性値につける結合キーが不要な分、若干格納効率が良い。

総合的なDIBの格納効率は1)と2)をあわせたものであり、テストデータで0.6程度であった。これから本手法の非正規形データ構造による格納効率の低下程度はそれほど悪くなく、検索速度の向上の方が利点が大きいと考えられる。

7.3ディレクトリスキーマの変更に対する対応性

ディレクトリスキーマやRDBスキーマのデータ構造をデータとして定義し、プログラム構造から分離したことで、容易にディレクトリスキーマの変更が行えた。ディレクトリスキーマは4種類の定義から構成されるが、そのうち属性のシンタックスを除くものの変更についてはデータの変更のみで可能であった。例えば新しいオブジェクトクラス"KDD社員"とか新しい属性タイプ"社内内線電話番号"といった要素の追加やDITでのオブジェクトクラス間のとりえる上下関係は自由に変更できる。属性シンタックスの変更については、新しいASN.1データ構造に対応するASN.1デコーダ/エンコーダの部分をプログラムに若干追加することで実現できた。また、各エントリテーブルの属性値用カラム数の変更といった非正規形の構造を最適化することも、データ変更のみで行えた。

8. おわりに

本稿では、汎用のリレーショナル型データベース管理システム(RDB)を用いたOSIディレクトリシステムのディレクトリ情報ベース(DIB)の実現方法について報告した。木構造をとるDIBを表現するのに非正規形のRDBデータ構造を用いることで検索速度の向上を図った。また、DIB構造及びRDBのデータ構造を可能な限りプログラム構造から分離し、DIB構造の変更に対処可能とした。評価の結果、データ件数の増加に対しても検索速度はあまり低下せず、比較的大規模のDIBが構築可能なことを実証した。今後は、同様なデータ構造を持つ他の応用(OSI管理など)への本手法の適用について検討を進めていく予定である。

最後に日頃御指導頂くKDD上福岡研究所 小野所長、浦野次長、鈴木コンピュータ通信研究室長に感謝します。

参考文献

[1]: 小花,西山,鈴木,"OSIディレクトリシステムの実装と評価",情処マルチメディア通信と分散処理研究会42-11,1989

[2]: 小花,西山,"OSIディレクトリシステムの実装(1) - 基本設計 -",第36回情処全大,1988

[3]: 西山,小花,"OSIディレクトリシステムの実装(2) - 汎用RDBパッケージを用いたDSA機能の実現 -",第36回情処全大,1988.

[4]: 西山,中尾,小花,"ディレクトリシステムのデータ構造に関する一考察",第34回情処全大,1987.

[5]: 堀内,西山,小花,"関係データベースを用いたOSIディレクトリのディレクトリスキーマの実現",第38回情処全大,1989.

[6]: 小花,吉満,西山,"OSIディレクトリシステムの標準化動向",国際通信の研究,1987

[7]: CCITT勧告X.500シリーズ/ISO IS9594(ディレクトリシステム)

[8]: CCITT勧告(1988年版)X.400シリーズ(メッセージ通信処理システム)