異種分散情報源の統合利用に対する ODMG OQL による問合せの実現

佐 藤 秀 樹

概 要

本論文では，インターネットを代表とするコンピュータネットワーク上に分散して存在する XML，SGML，HTML 等で記述された構造化文書ファイルとオブジェクトデータベースとの異種分散情報源に対する統合利用方式を提案する。このために，我々が研究・開発中のオブジェクトデータベースシステム「出世魚」を適用する。この方式は，(1) 分散共有仮想記憶によるデータの分散性の解消，(2) ラージオブジェクト機能による構造化文書ファイルの物理的異種性の解消，(3) マルチタイプオブジェクト機能による構造化文書テキストの論理的異種性の解消，(4) Object Database Management Group (ODMG) 標準の Object Query Language (OQL) による問合せ機能，により簡便に実現される。特に，ODMG OQL による問合せ機能は，オブジェクトデータベース外のデータも含めた異種分散情報源の統合利用に対して標準的な高水準問合せ言語を使用可能とし，Web 情報システムを始めとするアプリケーションの開発効率を向上させる。

1 まえがき

Web はコンピュータネットワークを介してアクセス可能なデータ空間であり，人間の知識を具現化する。このデータ空間の実現に向けた Web 情報システム技術 [1] の進展に伴い，近年，Web は益々広がってきている。こうした Web 情報システムはインターネットを代表とするコンピュータネットワーク上に構築され，ネットワーク上に分散して存在する情報源を利用することを特徴とする。また，これらの情報源はオブジェクトデータベース，関係データベース，構造化文書リポジトリなどの多岐に渡ることも特徴になっている。従って，コンピュータネットワークによる分散システム環境における異種情報源の統合利用への要求が高まってきている。

異種分散情報源の統合利用を行うためには，以下の課題の克服が求められる。

分散性：利用者・応用プログラムは，コンピュータネットワーク上に分散されたデータを利用することができる。

異種性：利用者・応用プログラムは，異なる形態で管理されているデータを統合的に利用することができる。

問合せ：利用者・応用プログラムは，多用なデータを柔軟に検索，抽出，再構成することができる。

本論文では，電子図書館，WWW (World Wide Web)，CALS 等の幅広い応用で利用される XML (eXtensible Markup Language) [2]，SGML (Standard Generalized Markup Language) [3]，HTML (Hyper Text Markup Language) 等で記述された構造化文書ファイルとオブジェクトデータベースとの異種性を想定する。そして，我々が研究・開発中のオブジェクトデータベースシステム「出世魚」を適用した異種分散情報源の統合利用方式を提案する。この方式は，(1) 分散共有仮想記憶 (Distributed Shared Virtual Memory; DVSM) [4] によるデータの分散性の解消，(2) ラージオブジェクト機能 [5] による構造化文書ファイルの物理的異種性の解消，(3) マルチタイプオブジェクト機能 [6] による構造化文書テキストの論理的異種性の解消，(4) Object Database Management Group (ODMG) 標準の Object Query Language (OQL) [7] による問合せ機能，により簡便に実現される。特に，ODMG OQL による問合せ機能は，オブジェクトデータベース外のデータも含めた異種分散情報源の統合利用に対して標準的な高水準問合せ言語を使用可能とし，Web 情報システムを始めとするアプリケーションの開発効率を向上させる。このため，異種分散情報源に対する統合利用方式の中核を成す，OQL コンパイラと C++OQL [7] の実装を中心に述べる。

本論文の構成は，以下の通りである。**第2節**では，「出世魚」のシステム構成と DVSM，ラージオブジェクト機能，マルチタイプオブジェクト機能を概説する。**第3節**では，異種分散情報源の統合利用方式を示す。**第4節**では OQL コマンドの翻訳処理を，**第5節**では C++OQL を，各々実装の観点から述べる。**第6節**では関連研究に言及し，**第7節**では本論文のまとめと今後の課題を示す。

2　「出世魚」のシステム概要

「出世魚」はオブジェクトデータベースシステムであり，ODMG3.0 [7] に準拠している。**図1**は，「出世魚」のモジュール構成を示す。「出世魚」はネットワーク上での分散並列機能の導入により，規模拡張性を実現している。システムは，WAKASHI [4]，INADA，WARASA から成る。WAKASHI はデータモデル独立であり，永続データ記憶を提供する。これは排他制御やリカバリといった基本的なトランザクション管理機能をサポートする。INADA は，ODMG3.0 C++Binding [7] に準拠するオブジェクトプログラミング言語を提供する。一方，WARASA は ODMG3.0 に準拠する OQL [7] をサポートする。以下では，「出世魚」の機能の

中より，異種分散情報源の統合利用方式を実現するDSVM [4]，ラージオブジェクト機能 [5]，マルチタイプオブジェクト機能 [6] について簡単に述べる。

分散共有仮想記憶：WAKASHIは，データベースファイル上にマップされるDVSMをサポートする。これは，複数の計算機に分散格納されたデータベースを１つのデータベースであるかのように扱うことを可能とする。

ラージオブジェクト：INADAは，既存ファイルを分散共有仮想記憶上にマップさせ，その領域にネットワーク上で一意となる識別子を割当てることにより，ファイルに格納されるデータを永続オブジェクトとして扱うことを可能とする。マップされた領域はラージオブジェクトと呼ばれ，その識別子は他の永続オブジェクトの識別子と同様に扱われる。

マルチタイプオブジェクト：マルチタイプオブジェクト機能はINADA独自の拡張機能であり，ODMG3.0 C++Binding [7] に準拠してはいない。これは，オブジェクトに対して動的に型の追加・削除[1]を行うことを可能とする。

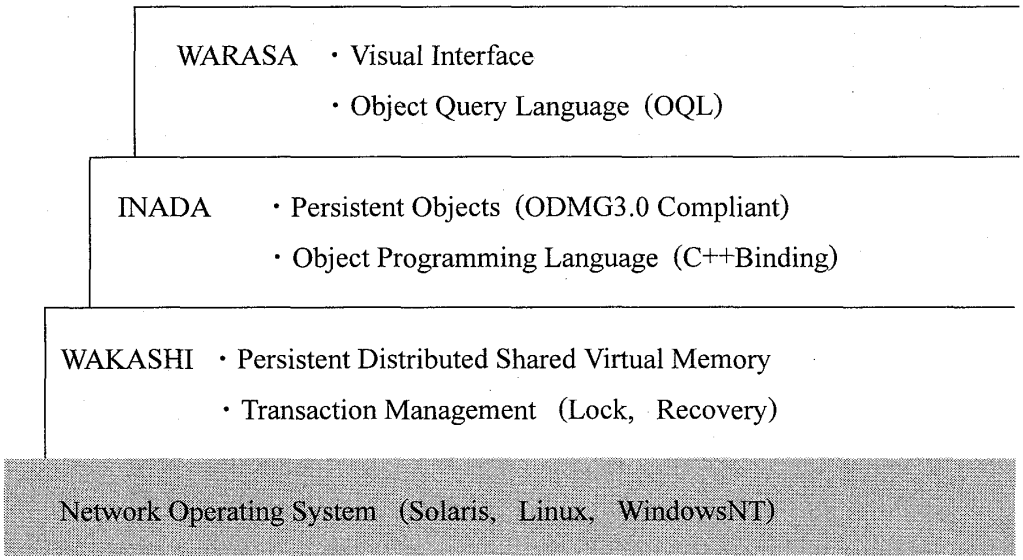

図１ 「出世魚」のモジュール構成.

３ 「出世魚」による異種分散情報源の統合利用方式

図２に，オブジェクトデータベースと構造化文書リポジトリの「出世魚」による統合利用方式を示す。オブジェクトデータベース（サイト１）に研究者オブジェクト“Researcher”が格納されている。一方，構造化文書リポジトリ（サイト２）には，例えばXML形式の論文情報“Paper”が格納されているとする。このとき，データベースを専門分野としている研究者情報と，その研究者によって書かれた論文のアブストラクトをまとめた構造化文書を作成

することを仮定する。このデータ操作要求を実現するため，**図2**の方式では，統合利用のための課題が以下のように解決される。

分散性の解消：DVSMにより，サイト1とサイト2では同じ記憶空間がサポートされる。

物理的異種性の解消：論文情報“Paper”を格納するXMLファイルは，ラージオブジェクト機能により，データベースファイルと同様にDVSMに動的にマップされる。

論理的異種性の解消：XML文書の構造を定義する型StructuredPaper[2)]を定義し，これをマルチタイプオブジェクト機能によりXMLファイルに格納されるデータに付加する。この型を通すことにより，XMLファイルに格納されるテキストデータを複合オブジェクトとして扱うことが可能となる。

問合せの実現：OQLにより，研究者オブジェクトと論文情報に対する検索が行われる。

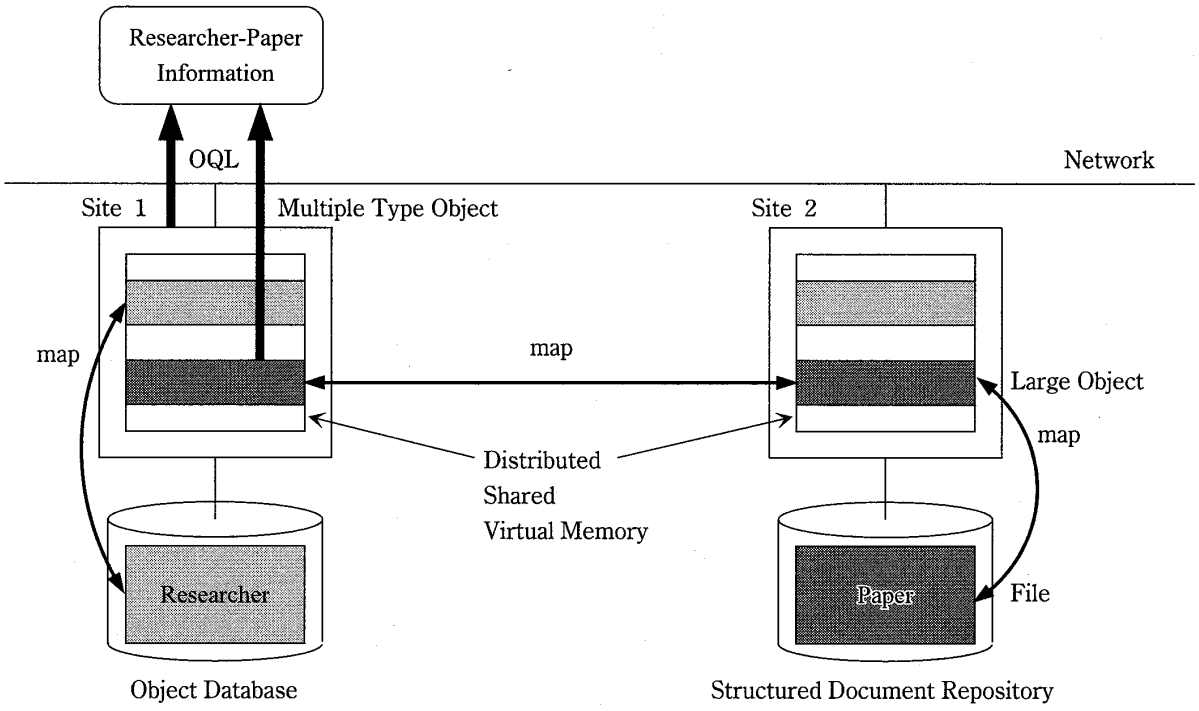

図2　異種分散情報源の統合アーキテクチャ

図3は，上記のOQLコマンドを示す。問合せにおけるas（“StructuredPaper”）は永続ポインタクラスd_Ref<T> [7]のメンバ関数の呼出しであり，型StructuredPaperを通して永続オブジェクトをアクセスする。このメソッドはパス式において通常のメソッドと同様に使用可能であり，OQL仕様の拡張を必要としない。論文情報“Paper”はXML形式のテキストデータであるが，ビュー（型）StructuredPaperを通すことにより論理的な構造が抽出され，複合オブジェクトとしてアクセスされる。そして，このオブジェクトと研究者オブジェクト“Researcher”とを使った問合せがOQLにより記述されている。

```
select struct(r, p.as("StructuredPaper").abstract())
from Researchers r, Papers p,
    p.as("StructuredPaper").authors() a
where r.field="database" and r.name=a.getElement()
```

図３　OQL. コマンド例

4　OQL コマンドの翻訳処理

OQL コンパイラは，OQL コマンドを C++OML プログラムに翻訳する。**図４**は，OQL コマンドの翻訳処理過程を示す。パーサーは OQL コマンド文字列を入力とし，パーサー木を出力する。パーサーは，字句解析器，構文解析器，意味解析器から成る。字句解析器は，OQL コマンド文字列からトークン列を生成する。構文解析器は，トークン列から構文解析木を生成する。意味解析器は，構文解析木をパーサー木に変換する。意味解析の処理では，OQL コマンドに現れる識別子／リテラルが識別子表に登録され，パーサー木より識別子表の対応するエントリがリンクされる。また，OQL コマンドに現れる識別子／リテラルに関する型チェックが行われる。構文解析器と字句解析器は，Yacc [8] / Lex [9] を使って実装されている。

変換器はパーサー木を入力とし，問合せ木を出力する。問合せ木の表現は，オブジェクト代数に基づく。問合せ木の非葉ノードはオブジェクト代数のオペレータに，その子ノードはオペランドに対応する。また，問合せ木の葉ノードは識別子表のエントリにリンクされ，対応する識別子／リテラルを表わす。問合せ木の非葉ノードには，対応するオペレータの実行結果の型が子ノードの型に基づき再帰的に求められる。

最適化器は問合せ木を入力とし，これを最適化した問合せ木を出力する。最適化器は，問合せ木標準化，問合せ木分解，選択オペレータ移動，オペレータ評価法決定，アクセスパス

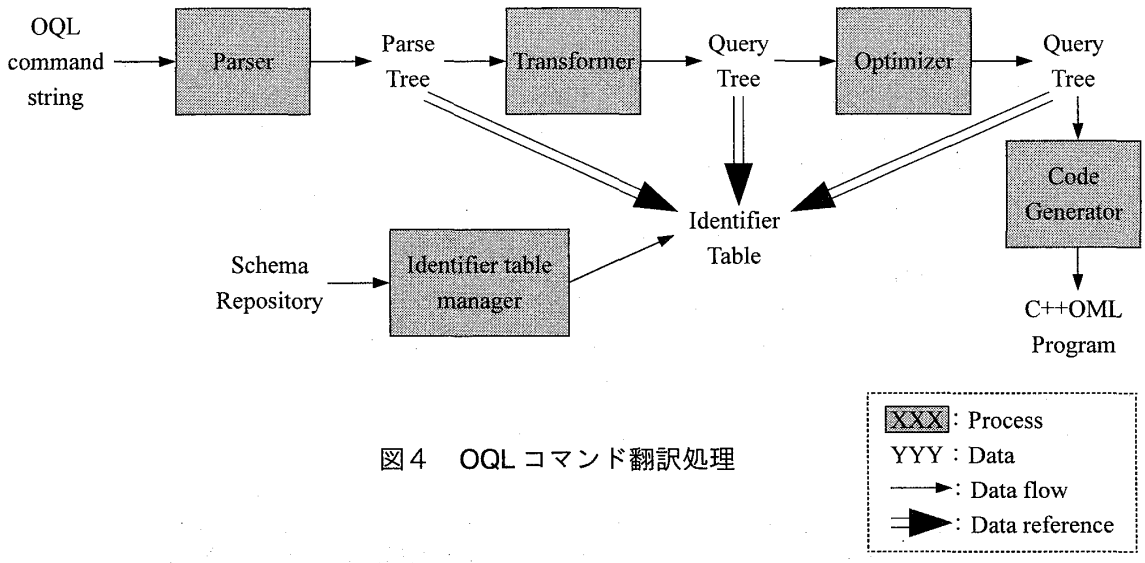

図４　OQL コマンド翻訳処理

選択といった処理から成る。問合せ木標準化は，問合せ木の問合せ条件を選言標準形に変換する。問合せ木分解は選言標準形となった問合せ条件の各選言項単位に元の問合せ木を分解し，部分問合せ木の集合を作成する。元の問合せ木は，各部分問合せ木をオペランドとする和集合オペレータを根ノードとする問合せ木に等価となる。各部分問合せ木の問合せ条件は連言形式となっており，この形式を仮定して以降の処理が行われる。選択オペレータ移動は，選択オペレータができる限り早く評価されるように，問合せ木内でそれを下方に移動する。これは，後に行われる結合オペレータ等の評価コストの低減につながる。オペレータ評価法決定は，問合せ木中の各オペレータにコスト予測に基づき評価アルゴリズムを割り当てる。特に，結合オペレータの評価法 [10]，パス式の評価法 [11] 等は潜在的に高コストであり，評価アルゴリズムは適切に決定される必要がある。アクセスパス選択は，問合せ木中の各オペレータ評価に際して，オペランドとなるオブジェクトへのアクセス法を決定する。例えば，エクステントに対するインデックスが編成されていれば，その利用の可否が検討される。

コード生成器は問合せ木を入力とし，C++OML プログラムを出力する。**図5**は，**図3**の

```
d_Bag<struct<d_Ref<Researcher>,d_Ref<XML_Element>>> result;
d_Bag<d_Ref<Researcher>>result1=
   new(d_Database::transient_memory,"d_Bag<Researcher>")d_Bag<Researcher>;
d_Iterator<d_Ref<Researcher>>iter1, iter2;
d_Iterator<d_Ref<Paper>>iter3;
d_Iterator<d_Ref<XML_Element>> iter4;
d_Ref<Researcher> p1, p2;
d_Ref<Paper> p3;
d_Ref<XML_Element> p4;

iter1=Researchers.create_iterator();
while(iter1.next(p1)){
 if(p1.field=="database"){
  result1->insert_element(p1);
 }
}
iter2=result1.create_iterator();
while(iter2.next(p2)){
 iter3=Papers.create_iterator();
 while(iter3.next(p3)){
  iter4=p3.as("StructuredPaper").authors().create_iterator();
  while(iter4.next(p4)){
   if(p2.name==p4.getElement()){
    result->insert_element(struct(p2,p3.as("StructuredPaper").abstract()));
   }
  }
 }
}
```

図5　OQL コマンドに対して翻訳された C++OML プログラム断片.

OQLコマンドに対して出力されるC++OMLプログラム断片の例である。簡単のため，結合オペレーションはNested-Loopアルゴリズムにより評価される。また，各エクステント等には，インデックス等のオブジェクトへの高速アクセス手段は用意されていないこととする。

識別子表管理は，OQLコマンドに現れる識別子／リテラルに関する定義情報等を管理する。これらは，OQLコマンド内で宣言されるもの，データベース内のスキーマリポジトリに定義情報が格納されるものに分かれる。また，識別子／リテラルは，OQLコマンド内での有効範囲を有する。従って，識別子表は有効範囲を反映するデータ構造となっている。パーサー木あるいは問合せ木の葉ノードからは，関連する識別子／リテラルに対応する識別子表のエントリがリンク付けされる。

5　C++OQLの実装

C++OQLは，C++Binding上にマップされたOQLである。C++OQLは，Web情報システムを始めとする応用プログラムの開発効率向上をもたらすことが期待される。**図6**は，C++OQLを含むC++Bindingプログラムの一部を示す。図中の(a)はOQLコマンドを表わすd_OQL_Query型[7]の変数宣言文であり，変数qには初期的にOQLコマンド文字列が示すd_OQL_Queryオブジェクトが代入される。OQLコマンド中の“$1”はパラメータ変数であり，(b)の実行文によりパラメータ値が結合される。(c)はOQLコマンドの実行文であり，実行結果は変数researchersに返る[3)]。

C++OQLを含むC++Bindingソースプログラムは，OQLコンパイラにより C++OQLを含まないC++Bindingオブジェクトプログラムに変換される。OQLコンパイラはC++OQLを含むC++Bindingプログラムのコントロールあるいはデータのフロー解析などを行うことなく，単に局所的な置換の機械的な繰返しによりこの変換を行う。**図7**のC++Bindingプログラムは，図6のC++Bindingソースプログラムに対するOQLコンパイラによる変換結果である。図中

```
d_Bag<d_Ref<Researcher>> researchers;
d_Iterator<d_Ref<Researcher>> iter;
d_Ref<Researcgher>> p;
d_OQL_Query q(                                  // (a)
  "select r from Researchers r where r.field=$1");
q<<"database";                                  // (b)
d_oql_execute(q, researchers);                  // (c)
iter=researchers->create_iterarot();
while(iter.next(p))
  cout<<p->name()<<endl;
```

図6　C++OQLを伴うC++Bindingプログラム断片

```
#include "d_OQL_Query_Defs.h"
d_Bag<d_Ref<Researcher>> researchers;
d_Iterator<d_Ref<Researcher>> iter;
d_Ref<Researcgher>> p;
d_OQL_Query1 q;                    // (a)
q<<"database";                     // (b)
(q).execute(researchers);          // (c)
iter=researchers->create_iterarot();
while(iter.next(p))
  cout<<p->name()<<endl;
```

図7　C++OQL. を伴わない C++Binding プログラム断片

の網かけ部分は，ソースプログラムからの変更を示す。**図7**の (a) は，変数 q の型がソースプログラムにおける d_OQL_Query から d_OQL_Query1 に変更される。d_OQL_Query1[4] は**図6**の (a) の OQL コマンドに対応する d_OQL_Query の下位型であり，OQL コマンドの実行コードは d_OQL_Query1 の execute メンバ関数内に展開される。この execute 関数は d_OQL_Query では仮想メンバ関数として宣言されるため，同一の execute 構文により異なる OQL コマンドのメソッド呼出しが可能となる。**図7**の (c) では，ソースプログラムにおける関数 d_oql_execute の呼出しがメンバ関数 execute の呼出しに変更されている。この実装は，オブジェクトモデルの特徴であるポリモロフィズムを利用している。また，OQL コマンドのパラメータ変数に値を結合する文は，ソースプログラムのままで変更されない（**図7**の (b) 参照）。

OQL コンパイラにより，**図7**の先頭行に挿入される#include 文が示すヘッダファイルには，d_OQL_Query ならびにその下位型（i.e.，上記の d_OQL_Query1）の定義が含まれる。**図8**は，OQL コマンドのパラメータ変数に結合される値を保持する構造体型 parameter の定義を示す。フィールド parameter_type は，パラメータ変数のデータ型を示すコードを保持する。フィールド parameter_value は共用体であり，parameter_type が示すデータ型に応じてパラメー

```
struct parameter{
  int parameter_type;
  union{
    char* cvalue;
    d_String* dSvalue;
    d_Char dCvalue;
      ……
  }parameter_value;
};
```

図8　パラメータ結合情報を格納するデータ構造

タ値を保持する。**図9**は，パラメータ変数に値を結合するd_OQL_Queryの<<演算子の定義例を示す。パラメータ変数に結合される値（演算子の第2引数）のデータ型に応じて，<<演算子は多重定義となる。図中の(b)の注釈が示すように，parameterListはd_OQL_Queryのメンバ変数であり，パラメータ変数に結合される値情報を保持するparameter型の構造体を要素とするリストである[5]。

図10はd_OQL_Query型における execute 仮想メンバ関数を示す。各関数は，OQLコマンドの実行結果の型に応じた多重定義となっている。**図11**は，d_OQL_Query型の下位型であるd_OQL_Query1（**図7**参照）の概略を示す。図の executeメンバ関数内の(a)には，パラメータ変数に結合された値を代入・参照するための変数のリストが宣言される。(b)では，(i) OQLコマンド中のパラメータ変数の数とd_OQL_QueryのparameterListメンバの要素数とが一致するか否か，(ii) 各パラメータ変数と結合された値のデータ型に互換性があるか否かが検査され，(a)で宣言される変数にparameterList中の対応する要素から値が代入される。(c)には，**図5**と同様にOQLコマンドに対するC++OMLコードが展開される。この実行コードは，アドホックなOQLコマンドに対する実行コードに基本的に同じである。唯一の相違点は，OQLコマンドのパラメータ値が(a)で宣言される変数を介して参照されることにある。(d)では，上記の実行時に発生する誤り・例外事象に対する処理コードが展開される。C++OQLには，OQLコマンドの実行結果を歴訪子を介して返すインタフェースも用意されてい

```
friend d_OQL_Query& operator<<(d_OQL_Query  &q, const char *s){
  parameter* bound_parameter;
  bound_parameter=new parameter;
  bound_parameter->parameter_type=#cvcode;
                    // (a) #cvcode stands for character string type.
  bound_parameter->parameter_value.cvalue=s;
  q.parameterList.push_back(*bound_parameter);
   // (b) The parameterList is a list of parameter value binding information.
  return q;
}
```

図9　パラメータ値結合演算子の例

```
virtual void execute(char*){cerr<<"Invalid result type"<<endl;}
virtual void execute(d_String&){cerr<<"Invalid result type"<<endl;}
virtual void execute(d_Char&){cerr<<"Invalid result type"<<endl;}
 ..................
```

図10　d_OQL_Queryクラスの仮想executeメンバー関数

```
class d_OQL_Query1: public d_OQL_Query{
public:
 void execute(d_Bag<d_Ref<d_Object>> &result){
  (a)List of variable declaration for $i parameters.
  try{
   (b)Parameter examination and assignment to variables.
    (c)Execution codes of OQL command
      "select r from Researchers r where r.fild=$1".
  }
  catch{
   (d)Exception handling codes.
  }
 }
 void execute(d_Iterator<d_Ref<d_Object>> &result){
  (e)Codes in the case that the result is an iterator.
 }
}
```

図 11　d_OQL_Query 下位クラス定義の概要

る。(e) には，このための execute メンバ関数の実行コードが展開される。この実行コードは，上述した execute メンバ関数を呼出し，その実行結果に対する歴訪子を生成することにより実現される。

6　関連研究

異種分散情報源に対する統合利用方式の代表は，ラッパー・メディエータ・アーキテクチャである。例えば，DISCO [12] は複数データ源にアクセスする異種分散データベースであり，**図 12** に示すラッパー・メディエータ・アーキテクチャに基づく。DISCO のデータ源は，データベース，ファイル，専用データサーバー，HTML ページを含む。メディエータは SQL 風宣言的言語の問合せを各ラッパーに対する関係代数演算子の部分問合せに分解する。この過程において，メディエータは各データ源のデータモデル，スキーマ，問合せ言語の差異を解消する。また，ラッパーはデータ源との間でデータ変換を行うことにより，構造化ビューをサポートする。

「出世魚」では DSVM により分散性が解消されるのに対して，DISCO ではこの問題は各データ源に委ねられる。DISCO のラッパー機能は，「出世魚」ではマルチタイプオブジェクトによるビューにより実現されている。一方，DISCO のメディエータは各データ源を問合せ言語レベルでの統合を図り，このために SQL 風宣言的言語と関係代数演算子をサポートする。これに対して，「出世魚」ではラージオブジェクトによる記憶レベルでの統合が行われ，言語レベルでの統合は ODMG OQL によりそのまま達成される。換言すれば，「出世魚」によ

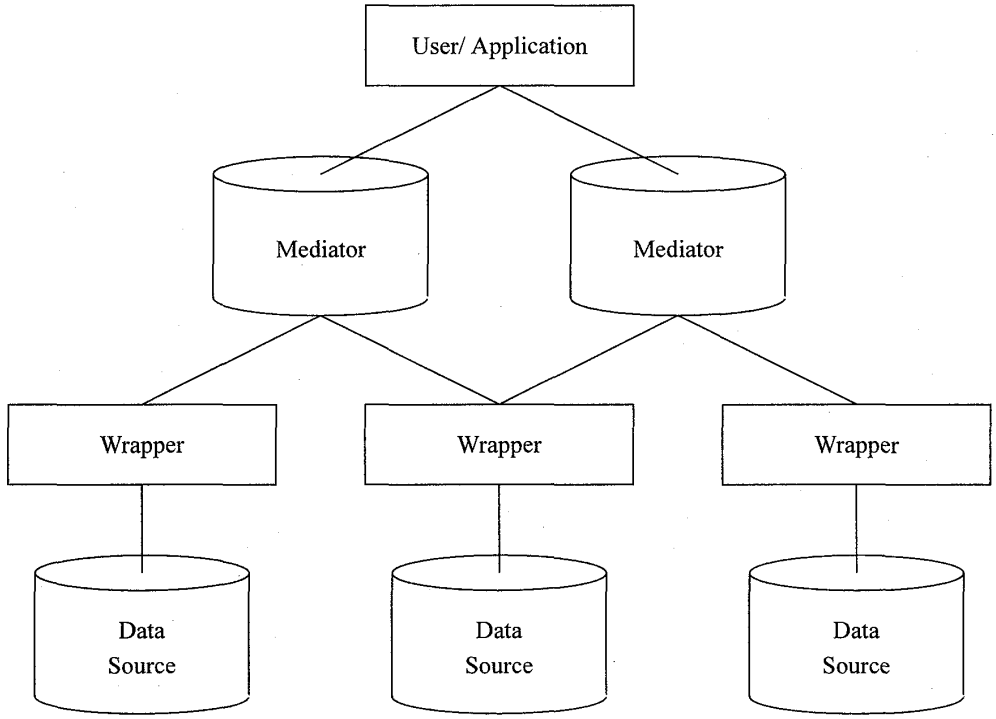

図 12　DISCO Wrapper-Mediator アーキテクチャ

る異種分散情報源に対する統合方式は，「出世魚」が元々備えている機能を使って実現されていることが特徴となっている。このことは，応用システムが「出世魚」により容易に実現できることを示している。

次に，C++OQL の実現に関して，関係データベースシステムにおけるコンパイル方式による親言語インタフェースの実現と比較する。System R [13] では，親言語 PL/I あるいは COBOL のプログラム内でのデータ準言語 SQL のコマンド埋込めが可能である。しかし，データ準言語の文は“$”で始まることになっており，親言語の文とは明確に区別される。また，SQL コマンドへのパラメータ受け渡しには親言語プログラムの変数が使われており，値の結合は暗黙的に行われる。さらに，いわゆるインピーダンスミスマッチの問題も抱えている。これに対して，C++OQL はデータベースプログラミング言語 INADA において使用可能であり，上記の言語仕様上の問題は存在しない。また，OQL コマンドへのパラメータ受け渡しは，オブジェクト操作により明示的に行われる。前者のコンパイルコードは，データベースアクセスモジュールの親言語の呼出し (CALL) 文により実現されている。一方，後者のコンパイルコードはオブジェクトモデルの特徴の一つであるポリモロフィズム，すなわち仮想メンバ関数を使ってエレガントに実現されている。

7　むすび

本論文では，XML，SGML，HTML 等で記述された構造化文書ファイルとオブジェクトデータベースとの異種分散情報源に対する統合利用を図る，オブジェクトデータベースシステム「出世魚」を用いた方式を提案した。この方式は，(1) 分散共有仮想記憶によるデータの分散性の解消，(2) ラージオブジェクト機能による構造化文書ファイルの物理的異種性の解消，(3) マルチタイプオブジェクト機能による構造化文書テキストの論理的異種性の解消，(4) ODMG OQL による問合せ機能，により実現される。本提案方式の特徴は，以下の通りである。

・「出世魚」の DVSM，ラージオブジェクト機能，マルチタイプオブジェクト機能，OQL により，方式は簡便に実現できる。
・本方式では，構造化文書ファイルはデータベース格納空間外に残されたまま，動的にマップされるため，ブラウザによる閲覧などはそのまま可能である。
・マルチタイプオブジェクトによるビューはメンバ関数といった手続きにより実現されるため，ビューは強力となる。
・オブジェクトデータベース外のデータも含めた異種分散情報源に対して，標準的な高水準問合せ言語が使用可能となり，Web 情報システムを始めとするアプリケーションの開発効率を向上させる。
・C++OQL は，オブジェクトモデルの特徴の一つであるポリモロフィズムに基づき簡易に実現される。

本論文では，オブジェクトデータベースと構造化文書ファイルを異種分散情報源とした。関係データベースなどの他の情報源の統合利用に関しては未検討であり，今後の課題となる。但し，JPEG (Joint Picture Expert Group) ファイルあるいは MPEG (Moving Picture Expert Group) ファイルに格納されるマルチメディアデータに関しては，格納構造が構造化文書ファイルと同様に原始的 (primitive) であり，かつデータ自体が自己完結的である。従って，本論文で提案した統合利用方式が，潜在的に適用可能でないかと期待している。その場合，マルチメディアデータに対するビューの実現にはマルチタイプオブジェクト機能が使われるが，メンバ関数といった手続きによる強力な記述力が有効となる。

注

1）オブジェクトに対する動的な型の追加・削除の詳細については，文献 6) を参照されたし。
2）構造化文書に対するビューの実装の詳細については，別の機会に報告を予定している。
3）実行結果に対する歴訪子を返すインタフェースも提供される。
4）ソースプログラム中の OQL コマンドには，各々を区別するように型名が付される。
5）パラメータ変数“$i”は，リスト parameterList の i 番目の要素に対応付けられる。

参 考 文 献

[1] D. Florescu, A. Levy, A. Mendelzon：Database Techniques for the World-Wide Web: A Survey, ACM SIGMOD Record, Vol. 27, No. 3, pp. 59-74 (1998).

[2] World Wide Web Consortium：Extensible Markup Language (XML) 1.0, *<http://www.w3.org/TR/1998/REC-xml-19980210>* (1998). W3C Recommendation 10-February-1998.

[3] ISO：ISO 8870：1986 Information Processing - Text and Office System - Standard Generalized Markup Language (SGML) (1986).

[4] G. Yu, K. Kaneko, G. Bai, and A. Makinouchi：Transaction Management for a Distributed Object Storage System WAKASHI - Design, Implementation and Performance, Proc. Int. Conference on Data Engineering, pp. 460-468 (1996).

[5] K. Kaneko and A. Makinouchi：Data Storage and Query Processing for Structured Document Database, Proc. Int. Workshop on Database and Expert Systems Applications (DEXA 97), pp. 92-97 (1997).

[6] M. Aritsugi and A. Makinouchi：Design and Implementation of Multiple Type Objects in a Persistent Programming Language, Proc. IEEE 19th Annual Int. Computer Software and Applications Conference (COMPSAC 95), pp. 70-76 (1995).

[7] R. G. G. Cattel, et al：The Object Data Standard：ODMG3.0, Morgan Kaufmann (2000).

[8] S. C. Johnson：Yacc：Yet Another Compiler-Compiler, Bell Laboratories (1979).

[9] M. E. Lesk and E. Schmidt, Lex - A Lexical Analyzer Generator, Bell Laboratories (1979).

[10] C. T. Yu and W. Meng, Priciples of Database Query Processing for Advanced Applications, Morgan Kaufmann (1998).

[11] Q. Fang, Q. Wang, G. Yu, K. Kaneko, and A. Makinouchi：Design and Performance Evaluation of Parallel Algorithms for Path Expression in Object Database Systems on NOW, Proc. DANTE'99, pp. 373-380 (1999).

[12] A. Tomasic, R. Amouroux, P. Bonnet, O. Kapitaskaia, H. Naacke, and L. Raschid：The Distributed Information Search Component (DISCO) and the World Wide Web, Proc. ACM SIGMOD'97, pp. 546-548 (1997).

[13] D. D. Chamberlin et al.：Support for Repetitive Transactions and Ad Hoc Queries in System R, ACM TODS, Vol. 6, No. 1, pp. 70-94 (1981).

謝辞 本論文は，九州大学大学院システム情報科学研究院・牧之内顕文教授との共同研究に基づくものである。この事を記して，感謝の意を表わしたいと思います。