データベースシステム 116－22
（1998. 7. 8）

OODBにおける拡張可能マルチメディアクラスライブラリの実装と評価

佐伯 剛幸　　波内 みさ

NEC C&Cメディア研究所

我々は、マルチメディアクラスライブラリを拡張性の高いフレームワークとして設計し、オブジェクト指向データベースPERCIO上で実装している。本マルチメディアクラスライブラリは、マルチメディアデータの格納、処理、および、検索に関して変更が予想される機能をカプセル化し、サブクラスにより再定義可能とすることにより拡張性を実現している。新規検索機構の組み込みの容易性を評価したところ、本クラスライブラリがソースコードの変更ではなく追加により拡張でき、追加するためのコード量を新規にライブラリを構築する場合に比べて大幅に削減できることを確認した。

Implementation and Evaluation of Extensible Multimedia Class Library for Object-Oriented Database Systems

Takayuki Saeki　　Misa Namiuchi

NEC C&C Media Research Laboratories

We have designed an extensible and customizable multimedia database class library using the concept of the object-oriented framework, and implemented it using our object-oriented database management system PERCIO. Its extensibility is achieved by encapsulating frequently modified functions into three submodules; media storage, retrieval and processing and by providing them as classes that can be extended by making subclasses. The results indicate that the class library is customizable by addition without modification to its source codes, and it can reduce the amount of source code for incorporating a new retrieval function.

1 はじめに

近年、マルチメディアの普及とソフトウェア部品化の流れにより、各データベースベンダは、データベース用のマルチメディア部品の整備を進めている[8]。これらのマルチメディア部品は、マルチメディアデータの格納、処理、検索などの機能をデータベースに追加するものである。リレーショナルデータベースベンダは、リレーショナルデータベースにオブジェクト指向的な機能拡張を施したオブジェクトリレーショナルデータベースの実現を行なっており、オブジェクト指向機能を基に部品を整備している。Informix/Illustra 社は Informix/Illustra データベースシステム用に DataBlade、Oracle 社は Oracle 用に DataCartridge、IBM 社は DB2 用に DB2 Extender というマルチメディア部品を製品化している。一方、オブジェクト指向データベースベンダである Object Design 社は、ObjectStore 用に Object Manager という名称でクラスライブラリ形態のマルチメディア部品を製品化している。

しかし、これらの既存製品では、部品中の拡張性が要求される構成モジュールが切り出されていないため、構成モジュールを入れ換えることにより容易に機能をカスタマイズすることができない。例えば、メディアコンテンツを管理する部品を考えると、イメージやテキストなどのメディア種別やデータサイズ、フォーマットなどに応じて管理方法を変更しなければならない可能性がある。拡張性を考慮したモジュールの切り出しを行なっていない部品では、このような機能拡張をすると部品全体の再構築が必要な場合が多い。

一方で、オブジェクト指向設計方法論の分野では、設計パターンやフレームワークなどの、クラス構造のレベルでソフトウェアの部品化を進め拡張性を高めるようなオブジェクト指向ソフトウェア再利用技術が提案されている [1][6][7]。

このような背景により、我々はオブジェクト指向設計技術を利用した拡張性の高いマルチメディア部品を実現することを目指し、拡張可能マルチメディアクラスライブラリの設計を行なっている[9]。本クラスライブラリは、フレームワークの考えに基づき、メディア格納機能、メディア検索機能、および、メディア処理機能をサブクラス定義により拡張可能であり、オブジェクト指向データベース PERCIO[11] 上で実装されている。

2 既存マルチメディア部品の課題と我々のアプローチ

2.1 既存マルチメディア部品の実現方法

オブジェクトリレーショナルデータベースのマルチメディア部品は、SQL3 のような宣言的問合せ言語からの利用 (のみ) が前提であり、新しいデータ型、そのデータ型のデータを処理する関数や演算子と高速アクセスのためのアクセスメソッド、関数から呼び出される外部モジュールから構成される。

オブジェクト指向データベースは、マルチメディア部品をクラスライブラリとして実現している。従って、利用言語 (例えば C++) からシームレスに扱うことができ、AP 記述能力が高く性能も良い。現在のところ SQL3 には対応していないため、追加された部品の問合せ処理最適化は行わない (利用者側で最適なメソッドを使用する必要がある)。

2.2 既存マルチメディア部品の課題

既存マルチメディア部品の主な課題として、以下をあげることができる。

1. 部品中の機能拡張性が要求される構成モジュールが切り出されていないため、機能拡張をするには部品全体の再構築と、データベースへ格納するためのデータ構造の変更が必要になることが多い。このため、以下の問題が発生する。
   (a) 再構築のための膨大な工数が発生
   (b) AP へ影響が波及
   再構築に対応して AP 側の修正が必要になる。
   (c) (データを格納済みの場合)DB スキーマの更新の問題が発生
   一般的に、データベースシステムでは、データを格納後はデータ構造は変更できないか、メンバ変数の追加などのごく小規模な範囲の変更のみ可能である。このため、データ格納後に構造を変更すると、変更前の構造から変更後の構造へのデータ変換が必要になる。
2. 同一機能を実現する複数の部品が存在し得るが、それらの API を統一し実現方法の入れ換えを可能とする枠組がない。例えば、テキスト全文検索に関して考えると、現在さまざまな検索機構を組み込んだ部品が独立して存在しているが、検索やインデックス生成などの API には互換性がない。

2.3 課題解決のためのアプローチ

上記の課題を解決するため、拡張が予想される機能をカプセル化し、サブクラスにより再定義可能とする。これにより、拡張可能な枠組みを提供する。拡張可能とする機能は、実装に直接関連する機能、検索機構などの他の技術を呼び出す機能などである。

3 拡張可能マルチメディアクラスライブラリ

本節では、マルチメディアクラスライブラリに要求される具体的な拡張性として、メディア格納機能、メディア検索機能、および、メディア処理機能の拡張性をあげ、それらを実現するためのアプローチの詳細をオブジェクトモデルと動作モデルにより示す。

3.1 メディア格納機能の拡張性

3.1.1 要求される拡張性

メディア格納機能では、コンテンツの格納場所とコンテンツサイズの多様性を吸収する拡張性を実現する必要がある。コンテンツの格納場所に関しては、ファイルで管理する場合、データベースに格納する場合、その両方に格納する場合などさまざまである。コンテンツサイズに関しては、サイズ大きなコンテンツを管理する場合には、分割してメモリへロードする必要がある。

3.1.2 アプローチ

前節であげた課題の解決方法をオブジェクト図と動作図を用いて説明する。設計のポイントを以下にまとめる。

1. メディア格納機能を実現するメディア格納クラスを導入し、メディアを抽象化するメディアクラスから格納機能を分離、利用者から格納機能の実装を隠蔽

2. メディア格納クラスへの操作を共通化することで多様な格納機能を統一、容易な格納機能の変更を実現

オブジェクトモデル

図1に、OMTの記法に従って記述した格納に関するオブジェクトモデルを示す。オブジェクト(インスタンス)は、このクラス構造に基づいてデータベースに格納される。メディアを抽象化するクラスとしてメディア(*Media*)クラスを導入し、それとは

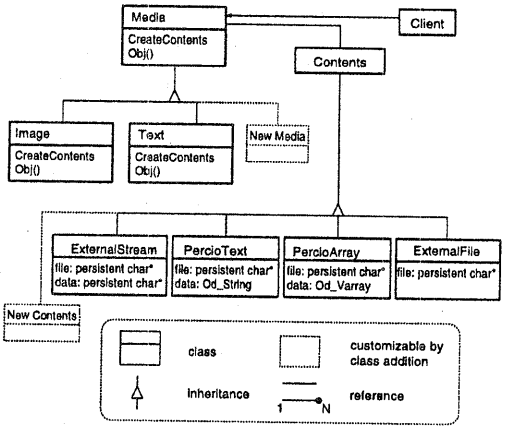

図 1: 格納に関するオブジェクトモデル

独立にメディア格納機能を実現するクラスとしてコンテンツ(*Contents*)クラスを導入している。

メディアオブジェクトは1つのコンテンツオブジェクトを参照し、メディアオブジェクトに対して要求されたコンテンツ関連の処理はコンテンツオブジェクトへ委譲される。利用者は、メディアオブジェクトのみを扱えばよく、それにつながっているコンテンツオブジェクトを直接扱う必要はない。このため、利用者に影響を与えずに、メディアオブジェクトから参照するコンテンツオブジェクト(つまり、格納機能)を変更することが可能になる。

コンテンツクラスのサブクラスである外部ファイル(*ExternalFile*)クラスは、コンテンツをファイルとして管理する場合に対応しており、図1に示すようにファイル名のみをメンバ変数 *file* で管理する。メンバ変数 *file* は persistent char* 型になっており、文字列へのポインタをPERCIOに格納することができる。一方、それ以外のクラスはコンテンツを格納するための領域をメンバ変数 *data* として持ち、コンテンツをデータベースに格納できる。メディアオブジェクトが参照するコンテンツオブジェクトを上記のサブクラスから選択することにより、利用者が要求する場所にコンテンツを格納できる。また、別の格納場所に保存したい場合には、コンテンツクラスのサブクラスを新しく定義し、そのオブジェクトをメディアから参照させる。これにより、メディアコンテンツのさまざまな格納場所への要求に対応できる。

図1に示すように、外部ファイルクラス以外のコンテンツクラスのサブクラスのうち、PERCIOテキスト(*PercioText*)クラスとPERCIO配列(*PercioArray*)クラスは、可変個の要素を収容するコレ

クションクラスを使用してコンテンツデータを分割して格納する。PERCIOテキストクラスは、メンバ変数dataの型としてPERCIOのOd_Stringクラスを利用し、テキストデータをPERCIOの可変長文字列として格納する。PERCIO配列クラスは、メンバ変数dataの型としてPERCIOのOd_Varrayクラスを利用し、バイナリデータをPERCIOの可変長配列として格納する。このため、これらのクラスでは、データベースから処理に必要な部分のコンテンツのみをロードできる。これに対して、外部文字列クラス(*ExternalStream*)ではコンテンツ全体をメモリにロードする。

メディアコンテンツを格納する場合に、PERCIOテキストクラス、PERCIO配列クラス、外部文字列クラスのうちのどのクラスのオブジェクトを生成しメディアオブジェクトから参照させるかは、メディアクラスの関数*CreateContentsObj*で制御しており、利用者に意識させずにコンテンツサイズに基づいて選択する。例えば、テキスト(*Text*)クラスでコンテンツを格納する場合、サイズがあらかじめ設定された値[1]よりも小さければコンテンツ全体をメモリにロードしても問題ないため外部文字列オブジェクトを生成する。大きければ一部のコンテンツのみをロード可能とするためPERCIOテキストオブジェクトを生成する。イメージ(*Image*)クラスの場合も同様である(ただし、PERCIO配列オブジェクトを生成する)。メディアクラスの各サブクラスでこのメソッドを再定義することにより、そのクラスごとに適したコンテンツオブジェクトを生成するように変更できる。これにより、メディアコンテンツサイズによる最適な管理方法の違いを吸収する。

動作モデル

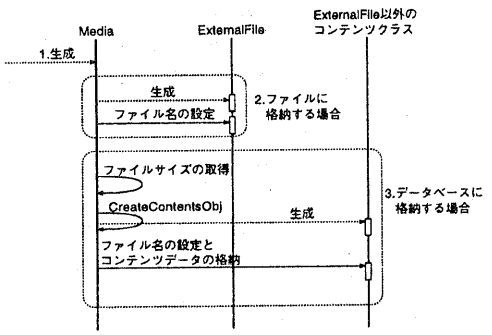

図2: 格納に関する動作モデル

ファイル名を指定してメディアオブジェクトを生成する場合の動作を、インタラクションダイアグラムの記法に従って記述した図2を用いて説明する。図2において、縦線がオブジェクト(クラス)を横線がオブジェクト間のメッセージのやりとりを示している。

メディアオブジェクト生成時に、利用者は同時にコンテンツ格納場所を指定する。ファイルに格納することが指定されると、外部ファイルオブジェクトを生成する(図2の2の場合)。データベースに格納することが指定されると、自分に対して関数*CreateContentsObj*を呼び出し適当なコンテンツオブジェクトを生成する(図2の3の場合)。

3.2 メディア検索機能の拡張性

3.2.1 要求される拡張性

マルチメディアデータに対する検索は、全文検索や類似画像検索などメディア固有の検索機構が存在し、さまざまな実現方式が検索エンジンとして実装されている。このため、メディア検索機能を実現する際には、以下に示す多様性を吸収する拡張性を実現する必要がある。

1. メディア固有の検索機構の存在
   全文検索や類似画像検索などメディアごとに固有の検索機構が存在するため、これらの検索機構のAPIを共通化することが望ましい。この理由として、APIが統一されていないと使いづらいこと、管理対象のメディアが増えた場合にそれに対応する検索機構を容易に組み込めないことなどをあげることができる。

2. 同種メディアに対するさまざまな検索機構の存在
   同種メディアに対しても、メディア検索機構は研究用から市販のものまで多数存在する。また、必要とされるすべての検索機構をあらかじめ予測することは困難である。このため、性能やコストなどにより最適な検索機構を選択できることが要求される。また、後から新しい検索機構を追加する場合でも、既存の検索機構に影響を与えずにそれらと併用できるようにする必要がある。

3.2.2 アプローチ

設計のポイントは以下のとおりである。

1. 検索機構を隠蔽するインデックスクラスを導入し、検索機構のAPIを共通化

2. メディア集合クラスを導入し、メディア集合に対する検索機構の選択をインデックスオブジェクトを入れ替えることにより可能とする

[1]利用者が、環境変数か環境設定ファイルで設定する。

オブジェクトモデル

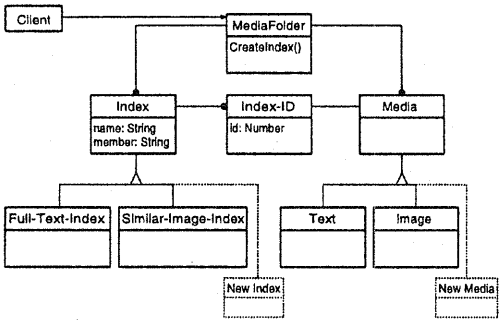

図 3: 検索に関するオブジェクトモデル

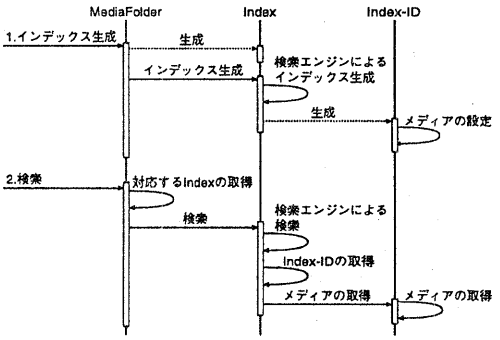

図 4: 検索に関する動作モデル

図 3に、検索に関するオブジェクトモデルを示す。インデックス (*Index*) クラスは、検索機構を隠蔽する。インデックスクラスのサブクラスとして、全文検索インデックス (*Full-Text-Index*) クラス、類似画像検索インデックス (*Similar-Image-Index*) クラスなどがある。これらのサブクラスは、インデックスクラスと共通のインタフェースを持つ。インデックス ID(*Index-ID*) クラスは、検索機構がメディアを識別するために生成する識別子 (ID) とメディアオブジェクトとの関係を管理する。これにより検索機構の検索結果に対応するメディアオブジェクトを取得することができる。

メディアフォルダ (*MediaFolder*) クラスは、メディアオブジェクトの集合を管理する。テキスト、イメージなど種別の異なる要素からなるメディア集合に対して、インデックスオブジェクトで隠蔽した検索機構を使用した検索ができる。利用者は、メディアフォルダのみを扱えばよく、インデックスオブジェクトを直接扱う必要はない。このため、利用者に影響を与えずに、インデックスオブジェクトの入れ替えで検索機構を変更することが可能になる。

動作モデル

図 4に動作モデルを示す。インデックス生成 (図 4の 1) では、生成対象として、メディアオブジェクトが保持するコンテンツデータ、ファイル名、または、メンバ変数名を指定する。生成対象と使用するインデックスの種別 (全文検索、類似画像検索など) を指定して、メディアフォルダオブジェクトに対して要求を出す。すると、種別に応じたインデックスクラスのオブジェクトが生成され、さらに、メディアフォルダに属するメディアの生成対象の値がインデックスオブジェクトを通して検索機構に渡され、インデックスが生成される。この時に検索機構が生成した ID とメディアオブジェクトとの関係を、インデックス ID クラスで管理する。

検索 (図 4の 2) では、インデックス生成時に指定した生成対象とインデックスの種別を指定して、メディアフォルダに対して要求を出す。メディアフォルダでは、指定に基づいて対応するインデックスオブジェクトを取得し、それを通して検索機構に検索処理を呼び出し結果の ID 集合を取得する。そして、インデックス ID オブジェクトを通して、ID に対応するメディアオブジェクトを取得する。

メディアフォルダが参照するインデックスオブジェクトを、全文検索インデックスオブジェクトや類似画像検索インデックスオブジェクトに切り換えることにより、適切な検索機構が呼び出される。さらに、インデックスクラスのサブクラスを新しく定義することにより、他の検索機構を取り込むことができる。これにより、メディア検索機能の拡張性を実現できる。

3.3 メディア処理機能の拡張性

通常、マルチメディアデータを処理 (表示、加工、編集、フォーマット変換など) するには専門の知識を必要とする。このため、メディア処理機能を、外部のメディア処理機構を組み込んで実現するがしばしば行なわれる。本マルチメディアクラスライブラリでは、ImageGear という市販のイメージ処理ライブラリ [12] を組み込んでいる。

3.3.1 要求される拡張性

メディア処理に関しては、さまざまなアルゴリズムおよび製品が存在する。このため、環境や要件などに基づいて最適なメディア処理機構の選択を可能とする拡張性を実現することが重要である。また、外部のメディア処理機構を取り込む場合には、バージョンアップなどによるインタフェースの変更など

へも対応できる必要がある。つまり、さまざまなメディア処理機構の容易な組み込みと入れ替えを可能とする枠組みを実現することが課題となる。

3.3.2 アプローチ

設計のポイントは以下のとおりである。

1. 利用者からメディア処理機能を隠蔽するメディア処理クラスを定義し、メディアクラスからメディア処理機能を分離する
2. メディア処理クラスのインタフェースを共通化することで処理機構を統一し、容易な処理機構の入れ替えを実現

オブジェクトモデル

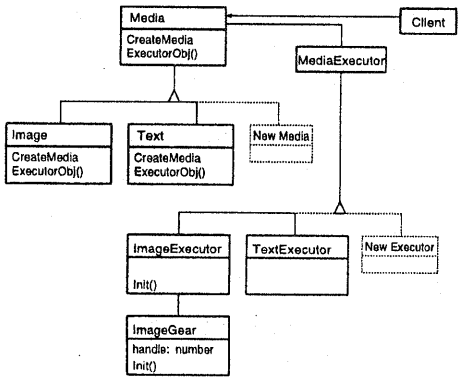

図 5: メディア処理に関するオブジェクトモデル

図 5に、メディア処理に関するオブジェクトモデルを示す。メディア処理機能を隠蔽するクラスとしてメディア処理 (*MediaExecutor*) クラス、イメージ処理 (*ImageExecutor*) クラス、テキスト処理 (*TextExecutor*) クラスを導入している。これらのクラスは抽象クラスであり、イメージ処理クラスのサブクラスとしてイメージギア (*ImageGear*) クラスが存在する。イメージギアクラスは、イメージ処理ライブラリ ImageGear を呼び出す。イメージギアクラスでは、ImageGear で管理するメモリデータを参照するためのハンドル値をメンバ変数で保持し、メディアデータ (メディアファイル名またはメディアコンテンツ) を ImageGear のメモリデータとして展開するメンバ関数 *Init* を提供している。イメージギアクラスにおける *Init* のような機構は、どのようなイメージ処理ライブラリでも必要になると考えられるため、イメージ処理クラスのメンバ関数として定義する。

メディアオブジェクトは 1 つのメディア処理オブジェクトを参照し、メディアオブジェクトに対して要求されたメディア処理はメディア処理オブジェクトへ委譲される。利用者は、メディアオブジェクトのみを扱えばよく、それにつながっているメディア処理オブジェクトを直接扱う必要はない。このため、利用者に影響を与えずに、メディア処理オブジェクトを入れ替えることが可能になる。

動作モデル

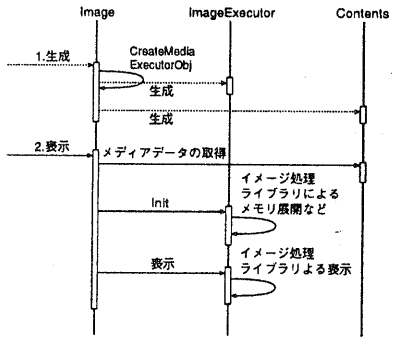

図 6: メディア処理に関する動作モデル

図 6に、イメージオブジェクトを生成しそれに対して表示処理を呼び出した時の動作モデルを示す。イメージオブジェクトは、生成時 (図 6の 1) に、自分自身に対して *CreateMediaExecutorObj* を呼び出す。*CreateMediaExecutorObj* は、あらかじめ設定された値[2]に基づいて、イメージ処理クラスのサブクラスのオブジェクトを生成する。

イメージの表示要求をイメージオブジェクトに対して行うと (図 6の 2)、イメージオブジェクトではコンテンツオブジェクトにメディアコンテンツの取得を要求する。イメージオブジェクトは、それを引数としてイメージ処理オブジェクトに対して *Init* を呼び出す。すると、イメージ処理ライブラリが、初期化処理として、受けとったコンテンツをメモリ上に展開する。イメージオブジェクトがイメージ処理オブジェクトに表示要求を出すと、イメージ処理ライブラリがメモリ上に展開済みのコンテンツを表示する。

利用者は、メディア処理クラスのサブクラスを定義し、このクラスのオブジェクトを生成するようにメディアのサブクラスで *CreateMediaExecutorObj* を再定義することにより、別のメディア処理機構を取り込むことができる。このように、メディア処理機構の容易な組み込みと入れ換えが可能である。

[2]環境変数か環境設定ファイルで設定する。

4 評価

本節では、本マルチメディアクラスライブラリをフレームワークとして構築することにより、従来のマルチメディア部品の課題が解決されるかを評価する。評価は、本クラスライブラリに検索機能として類似画像検索を追加することの容易性を調べることにより行なう。このため、全文検索機構のみが組み込まれている本クラスライブラリに対して類似画像検索機能を組み込む場合と、新規に類似画像検索機能を呼び出すライブラリを構築する場合を比較する。後者は、再構築する場合に相当する。

4.1 新規機構を組み込むための工数

本クラスライブラリに対して組み込む場合と、新規にライブラリを構築した場合の工数を比較するため、それぞれの場合のソースコード量を調査した。本クラスライブラリに組み込む場合では、図3に示したオブジェクトモデルに基づき、類似画像検索機構を隠蔽するSImage-Index、SImage-Index-IDというクラスを追加し、MediaFolderクラスを類似検索へ対応するため拡張した。新規にライブラリを構築する場合では、いくつかのクラスを定義した。IndexEngineクラスは、利用者からアクセスされるクラスであり、MediaFolderクラスに対応する。Indexクラスは、次に述べるIndex-IDクラスの集合を管理しており、SImage-Indexクラスに対応する。Index-IDクラスは、検索機構が生成する識別子とImageクラスオブジェクトの関係を管理しており、SImage-Index-IDクラスに対応する。IndexHandleクラスは、検索結果を管理する機能を持っており、本クラスライブラリではMediaFolderクラスで実現されている。

表1は、それぞれの場合のソースコードの量をクラスごとに示したものである。MediaFolderクラスに関しては、組み込むために追加したコード量と再利用できたコード量の両方を示している(括弧に囲まれた数値が再利用できたコード量である)。これから分かるように、本クラスライブラリに組み込む場合では、開発するソースコード量を54%削減することができた。

4.2 議論

表1においてクラスごとに比較してみると、本クラスライブラリのMediaFolderクラスに対して追加したソースコード量は、新規に作成した場合のIndexEngineとIndexHandleクラスのソースコードの合計量よりもずっと小さい。この理由は、本クラスライブラリにはメディア検索のための処理ロジックがすでに含まれており、類似画像検索で異なる部分のみを定義すればすんだためである。

工数以外の問題について考えると、クラスは変更ではなく追加されているので、アプリケーションへの影響はない。また、同じ理由によりデータの変換なども不要である。さらに、全分検索と類似画像検索という複数の検索機構を、実装の違いを意識せずに使用することができる。

5 関連研究・既存製品

5.1 データベース用マルチメディア部品

オブジェクトリレーショナルデータベース用の部品としては、Informix/IllustraのDataBlade、OracleのDataCartridge、IBMのDB2 Extenderなどがある。また、オブジェクト指向データベース上のクラスライブラリとして、ObjectStore用のObjectManagerがある。しかし、これらはいずれも拡張可能ではないと推定される。

5.2 マルチメディア情報管理フレームワーク

情報検索用フレームワーク(IR-FWK)[4]や、動画と音声という時間メディアを扱うフレームワーク(AV-FWK)[5]がある。

IR-FWKでは、インデックス、インデックス付加の対象となる属性、使用するインデックスを指定するパラメータなどをクラス化し、検索方法のカスタマイズが容易な構造を実現している。また、単一のメディア(テキスト)のみを検索の対象としており、異なるメディアを検索するための枠組はない。

これに対して、本マルチメディアクラスライブラリではインデックスのクラス化のみを行っており、属性やパラメータのクラス化までは行なっていない。このため、拡張性はIR-FWKより劣るものの、処理オーバーヘッドが少なくてすみ処理速度の面で有利である。また、異なるメディアを要素とする集合に対してインデックスを付加できる枠組を、メディアフォルダを導入することにより提供している。これにより、IR-FWKでは考慮されていない、異なるメディアに対する検索APIの共通化という課題を解決することができる。

AV-FWKでは、メディア処理機能を、メディアデータとそれを処理する装置という観点から抽象化したクラス構造を実現している。これに対して、本マルチメディアクラスライブラリでは、メディア処理クラスにより処理機能の抽象化は実現しているが、メディア処理クラスがメディアクラスと関連

表 1: 類似画像検索機構を組み込むのに必要なソースコードライン数

本クラスライブラリに組み込む場合		新規にライブラリを構築した場合	
MediaFolder	167 (2006)	IndexEngine	551
SImage-Index	496	Index	257
SImage-Index-ID	59	Index-ID	160
Image	—	Image	229
MediaFolder	—	IndexHandle	368
Total	722	Total	1565

付けられている。このため、メディアとメディア処理のモジュール分割が困難であるが、利用者からメディア処理を隠蔽することができる。これにより、利用者に意識させずに、クラスライブラリ内でメディア処理クラスを交換することにより最適なメディア処理機構を呼び出すことが可能になる。

6 まとめ

本研究では、メディア格納、メディア検索、および、メディア処理機能を拡張可能なマルチメディアクラスライブラリを実現するため、拡張性が必要と予想される機能をクラスとしてカプセル化し、サブクラス定義により拡張可能とした。これにより、検索機能を新規に追加するためのソースコード量を削減することができ、データ構造の変更を回避し、APへの影響の波及とスキーマ更新の問題を回避することができる。また、同一機能を実現する複数の部品を、部品ごとの実装の違いを意識せずに使用することができる。

参考文献

[1] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994.

[2] I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard, "Object-Oriented Software Engineering - A Use Case Driven Approach", Addison-Wesley, 1992.

[3] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorenson, "Object-Oriented Modeling and Design", Prentice Hall, 1991.

[4] G. Sonnenberger and H. P. Frei, "Design of a Reusable IR Framework", ACM SIGIR'95, 1995.

[5] S. Gibbs, C. Breiteneder and D. Tsichritzis, "Audio/Video Databases: An Object-Oriented Approach", *International Conference on Data Engineering*, 1993.

[6] H. Schmid, "Creating the Architecture of a Manufacturing Framework by Design Patterns", ACM Conference on Object Oriented Programming Systems, Languages and Applications(OOPSLA), 1995.

[7] H. Hüni, R. Johnson, R. Engel, "A Framework for Network Protocol Software", ACM Conference on Object Oriented Programming Systems, Languages and Applications(OOPSLA), 1995.

[8] M. Stonebraker, Object-Relational DBMSs, Morgan Kaufmann, 1996.

[9] 佐伯, 波内, "OODB における拡張可能マルチメディアクラスライブラリ", 情報処理学会第 55 回全国大会, 1997.

[10] 波内, 鶴岡, "SGML/HTML 文書 DB におけるテキスト格納方式の提案", 情報処理学会第 54 回全国大会, 3-205, 1997.

[11] 鶴岡, 他, "オブジェクト指向データベース管理システム PERCIO の開発と今後の課題", 電子情報通信学会論文誌,D-I, Vol. J79-D-1, No.10, pp.587-596, Oct. 1996.

[12] イメージプロセシングソフトウェア開発キット ImageGear ユーザーズマニュアル, ライフボート, 1997.