分散環境における画像オブジェクトの版管理機構の実現

田幡 勝[†,††] 有次正義[†] 金森吉成[†]

ネットワーク技術の発展にともない，分散して存在する各種の画像データベースへのアクセスが可能となった．一般に，異種のプラットフォーム上に存在する各画像データベースを透過的に操作することや画像処理を各データベースに対して実現することは重要な課題である．さらに，画像処理結果の管理手段を実現することは難しい問題である．そこで，我々は分散環境において画像データベースに対する画像の版管理を提供する機構を実現する．この実装では，画像処理および版管理は各画像データベースと独立に開発され，それらをそれぞれのデータベースに対し利用可能にする．また，この実装は画像処理の結果だけでなく画像処理の履歴も管理する．このため，ネットワーク内の画像を操作するとき画像処理結果および履歴を再利用することが可能となる．本論文では，この版管理機構の実装方法と画像処理結果と画像処理手順の再利用について説明する．

Implementing Version Management Mechanism for Image Objects under Distributed Environment

Masaru Tabata,[†,††] Masayoshi Aritsugi[†] and Yoshinari Kanamori[†]

The recent advances of network technologies enable us to access a large number of images stored in various distributed databases. In general, it is not easy to transparently manipulate image databases running on various platforms, nor to support image processing that are available for them. In addition, supporting the management of images generated by image processing is a challenging issue. We implement a version management mechanism for images stored in databases in a distributed environment. In the implementation, developing image processing and version management is independent of each database, and these facilities are available for every database. This implementation manages not only results of image processing but also histories of image processing applied to images. Then, it can allow us to reuse both results and histories in manipulating images in a network. In this paper, we describe the implementation for the version management mechanism and reusing results and histories of image processing with it.

1. はじめに

画像データベースを扱うアプリケーションプログラムを作成するためには画像処理は必須である．例えば，医療画像内の腫瘍を鮮明にするといった操作やリモートセンシングの画像をカテゴリ分類するといった操作は，複数の画像処理を組合せて実現される．このような一連の画像処理は複雑なものとなり，画像処理の専門的な知識がないと画像処理の実行順序やその画像処理で用いるパラメータを与えることは困難である．したがって，あるユーザが画像処理を実行して得られた結果やその手順を他のユーザが再利用できるための機構を提供することは，画像データベースのアプリケーション開発にとって非常に有効である．この機構を実現するために，我々はこれまでに画像データベース内の画像に対し一連の画像処理を実行したときに生じる結果および画像処理履歴を画像オブジェクト（Image Object）の版として管理するためのモデルを提案し，その設計を行ってきた[2),11),19)]．版管理の目的をまとめると，以下の二つが挙げられる．

(1) 一連の画像処理過程における途中結果を版として管理し，特定の時点における画素値データを再利用する．

(2) ある画像に対して一連の画像処理を実行したときに得られた画像処理手順の履歴を再利用する．

これらの版管理を行うことによって，複数のユーザが画像データベースを利用するとき，あるユーザが得た画

† 群馬大学工学部情報工学科
Department of Computer Science, Gunma University
†† 日本学術振興会特別研究員
Research Fellow of the Japan Society for the Promotion of Science

像処理に関する知識を他のユーザが再利用することが可能になる．従来のCADやプログラミングコード等に関する版管理の研究では（1）の場合の特定の時点での版を再利用することに焦点がおかれている[10]．この視点から商用のオブジェクト指向データベースシステムは（1）の版管理機能を用意している．しかし，画像処理を対象とする場合には，単にある時点での画像データを保存するだけでなく，（2）の場合のように画像処理の実行順序，個々の画像処理に用いられたパラメータ等を保存する必要がある．すなわち，一般的な版管理の他に画像処理の手順を管理する機構が必要となる．

また，コンピュータ・ネットワークの発展にともない，各種のホスト上にこれまでに構築されてきた多様な画像データベース資産が利用可能となってきた．しかし，これらのデータベースは様々なプラットフォーム上で独立に構築されているため，これらを統合利用することは容易ではない．例えば，ネットワークによって相互接続された医療機関がそれぞれが保有する医療画像を共有し診断することを考えると，それぞれの医療機関で画像データベースを実装したデータベース管理システム，OS，プログラミング言語等の異種性のため，統合利用には非常に困難な作業を伴う．多大な努力を必要とする具体例として，これらのプラットフォーム毎に画像処理演算を実装することを挙げることができる．この画像処理プログラムの保守，更新は画像処理に関する専門的な知識を必要とするため，専門家が管理することが望ましい．したがって，画像処理プログラムを各データベース毎に開発するのではなく，画像データベースと独立なサイトで開発できなければならない．画像処理プログラムの場合と同様に，画像処理の結果および手順を管理する版管理機能も画像データベースと独立に開発を行い任意の画像データベースに対して利用が可能となるようにする必要がある．

このような異種分散環境下でのアプリケーション開発を容易にするために，オブジェクト指向の技術を適用するアプローチであるCORBA（Common Object Request Broker Architecture）がOMG（Object Management Group）によって提案されている[15),16)]．オブジェクト指向技術を用いることによってサーバ側の実装はカプセル化され，クライアント側に対しネットワーク上に分散した画像データベースの位置やリモートの画像データベースにアクセスするための低レベルの通信プロトコルなどに関する情報が隠蔽される．また，OSやプログラミング言語の違ったマルチベンダの環境で実装された各システムを一つのシステムとして扱うことを可能にする．そこで，我々はCORBAを用いて分散環境に存在する複数の画像データベースを用いることを可能にする機構を実現する．

本論文では，まず2章で我々と同様にCORBAを用いてデータの統合利用を行っている研究について示す．次に，3章で画像データベースのための版管理機構の構成を述べる．そして，4章ではこの版管理機構の実装方法を述べ，その性能評価と利用例を示す．最後に5章で本論文のまとめと今後の検討すべき課題を述べる．

2. 関連研究

最近，ネットワーク上に分散した構造化文書やWWW等の統合利用に関する研究[8),9)]が盛んに行われてきている．また，医療データベースや地理情報システムの分野で，これらの既存のデータを統合利用する方法が必要とされている．しかし，このようなデータの多くは，異なったシステム上で開発されているため，統合利用することは容易ではない．このような異種のシステム間での相互運用性（interoperability）を達成するために多くの分散オブジェクト技術が実現されている．我々はネットワーク上に分散した画像データベースを統合利用するためにその中の一つであるOMGのCORBAを採用した．CORBAによってネットワーク上に分散したデータベースを統合することを実現している研究[3),4),12),17),20),21)]は数多くある．**表1**にこれらの比較を示す．

CORBAでは，クライアント側からサーバ側のオブジェクトを操作するためにIDL（インタフェース定義言語，Interface Definition Language）を用いてインタフェースが定義される．データベースの統合利用に対し，このインタフェースを定義する方法として表1に示すように大きく分けて次の二つが存在する．

（1） データベースのAPI（Application Program Interface）に対してインタフェースを定義し，それを介してデータベースにアクセスする（データベース単位のインタフェース）．

（2） データベースに対するインタフェースに加えて，データベースに格納された永続オブジェクトに対してもインタフェースを定義する（オブジェクト単位のインタフェース）．

MINDのような汎用的なマルチデータベースシステムでは，頻繁にスキーマの新規の定義や更新が行われることが多い．このとき，それぞれのクラスに対しIDLを用いてインタフェースを定義するならば，スキーマ定義が変更される毎にインタフェースの定義も変更されるため，システムを再コンパイルする必要が生じる．従って，（1）のデータベースに対するインタフェースのみ

表1 CORBAによる統合システム
Table 1 Integrated systems with CORBA

研究	目的・用途	ORB	DBMS	インタフェース
FZI's GIS[12]	地理情報システム	ORBeline☆, Orbix	Oracle	データベース単位
GSOT[20]	地理情報システム	VisiBroker	Illustra	データベース単位
MIND[3]	マルチデータベース	ObjectBroker	Oracle, Sybase, MOOD	データベース単位
Blob Streaming framework[17]	医療画像	Orbix	OODB	オブジェクト単位
IOS[2),11),19]	画像データベース	Orbix, JavaIDL	ObjectStore	オブジェクト単位
Multiware[21]	マルチメディア文書	Orbix	ObjectStore	オブジェクト単位
TeleMed[4]	医療画像，治療履歴	Orbix, ORBeline	ObjectStore, mSQL	オブジェクト単位

を定義する方法が有効である．

一方，本研究のような画像データベースの場合，スキーマの更新はまれであるため上記のような問題は生じない．このため，本研究で扱う永続オブジェクトに対して，そのオブジェクトに対するインタフェースを定義し，クライアントからサーバ側に存在する永続オブジェクトに直接メッセージを送ることができるようにしている．そのオブジェクトを分散オブジェクトとしても扱うことが可能となり，システムの構築に対しより柔軟性が与えられる．

TeleMed[4] や Blob Streaming Framework[17] は本研究と同様に画像データベースの統合を目的としたものである．TeleMedは，画像処理演算をアプリケーション開発に対し提供しているが，画像処理結果の管理手段は提供していない．また，Blob Streaming Frameworkは医療機関内での画像データの高速な転送の実現に重点を置いており，画像処理を扱っていない．本研究は，画像データベースに対する版管理を提供する点がこれらの研究と異なっている．

3. 版管理機構のアーキテクチャ

我々の実現した版管理機構の構成を図1に示す．分散した画像データベースを扱うアプリケーションプログラムに対し，画像オブジェクトサーバ（Image Object Server, IOS）が版管理機能を提供する．このシステムでは，画像データベースに格納されている画像データはオブジェクトとして扱われる．このオブジェクトを画像データオブジェクト（Image Data Object, IDO）と呼ぶ．画像データオブジェクトサーバ（Image Data Object Server, IDOS）が異種の画像データベースに対するラッパーの役割を果たし，IDOを他のサイトからアクセスすることを可能にしている．

ある単一の画像処理演算を実行するオブジェクトが，画像処理オブジェクト（Image Processing Object, IPO）である．画像処理オブジェクトサーバ（Image Processing Object Server, IPOS）によってIPOによる画像処理演算が提供される．画像処理オブジェクトを画像データオブジェクトと独立なサーバ上におくことによって，任意の画像データベースに対して画像処理演算を提供することが可能となる．また，我々はIPOを複数の計算機上に分散させることによって画像処理演算を並列化し高速化を図ることについて研究を進めている[5]．

IDOSおよびIPOSはそれぞれネットワーク上に分散して実装され，アプリケーションプログラムは，これらのサーバ上に分散したオブジェクトの操作をIOS上の汎画像オブジェクト（Generic Image Object, GIO）を通して行う．また，各サーバにあるManagerオブジェクトによってサーバの終了，データベースのオープンおよびクローズの機能が提供される．アプリケーションプログラムがGIOに画像処理メッセージを送ったとき，GIOがさらにIPOへ画像処理メッセージを送る．このとき，アプリケーション側からは，GIOはメソッドとして画像処理，属性値として画素値データを保持する画像オブジェクト（Image Object）であるように見え，IDOやIPOがネットワーク上に分散した実装を意識する必要はない．そして，GIOは一連の画像処理を実行して生じた結果および画像処理履歴を版として保持し，それが版データベース（Version DB）に保存される．

分散した画像データベースの情報を隠蔽するため，この機構はアプリケーション層，画像オブジェクトサーバ層およびデータベース層の3階層から構成され，これらの階層間の通信はCORBAによって行われる．このような3階層の構造とすることによって，アプリケーションプログラムに対して，画像データベースと画像処理が分散した実装がIOSによって隠蔽される．アプリケーションプログラムは，その下の階層の分散した実装は意識せずに，単にIOSに対してメッセージを送信することによって画像処理の依頼が行える．このようにIOSが画像処理および版管理の機能を提供することによって，アプリケーションプログラムの開発を容易に行うことができる．付録にこれらのオブジェクトに対して定義したインタフェースを示す．本論文では，図1のQuery

☆ ORBelineはVisiBrokerに名称を変更

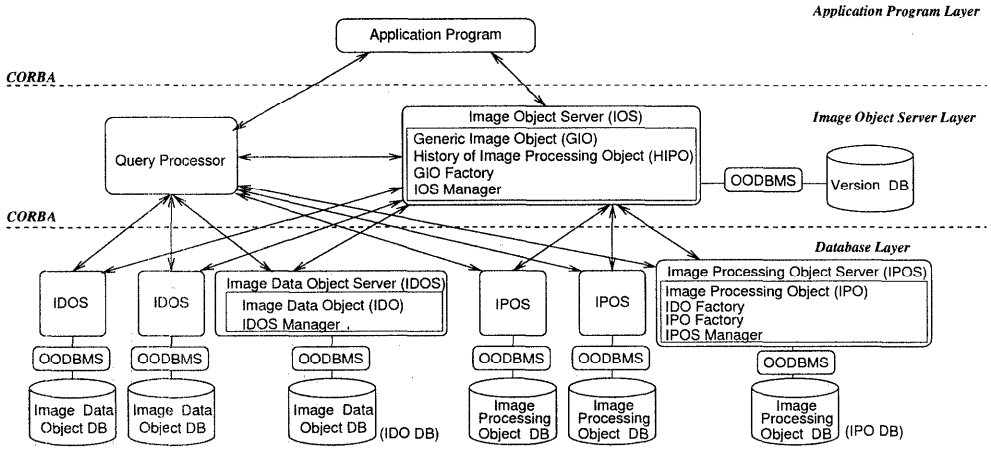


図 1 版管理機構のアーキテクチャ
Fig. 1 Version management architecture

Processor の説明は省略する.

我々の実装では，アプリケーションプログラムに対しては GUI の開発の容易さから Java を用い，IOS，IPOS および IDOS に対しては性能面から C++ をそれぞれ用いて開発している．ORB として Java IDL[18] および Orbix[7] を用い，オブジェクト指向データベースシステムとして ObjectStore[13] を用いて実装を行った．GIO，IDO および IPO は，CORBA によって提供される分散性とオブジェクト指向データベースシステムによって提供される永続性を併せ持っている．クライアント側からリクエストがあったとき，これらのオブジェクトは動的にデータベースからメモリ空間にロードされ[6]，リクエストされた処理を実行する．

今回の実装では，オブジェクト指向データベースを用いて IDO および IPO の作成を行ったが，既存の画像データファイルや画像処理ライブラリを IDO，IPO とすることも可能である．この場合，既存のライブラリを IDO および IPO のインタフェースがラッピングすることにより，既存の資産をオブジェクトとして扱うことが可能となる．

4. 版管理機構の実装

この章では，我々が実装を行った版管理機構について説明する．まず，版管理の機能を提供する汎画像オブジェクト（GIO）の構造について説明する．この GIO は，文献[11] の画像オブジェクトモデルに基づいて設計されている．次に，GIO を用いて画像処理を実行したときに生じる版を管理する方法とその版を再利用する方法について述べる．そして，実装システムの性能評価を行い，版管理機構を用いたアプリケーションの例を示す．

4.1 汎画像オブジェクトの構造

画像の版を管理するためには，ある特定の時点での画像処理結果を保存する必要がある．また，画像処理の手順を再利用するためには，ある版を生成したときの導出元の版がどれであるか，そのとき用いた画像処理は何か，その画像処理に用いたパラメータの値といった情報が必要である．一般的な商用のオブジェクト指向データベースは，特定の時点での版を管理する機能を提供するだけであるため，一般的な版管理の他に画像処理の手順も管理できる版管理機構が必要となる．このため，以下に説明するクラスを導入し，画像に対する版管理機能を提供する．

まず，個々の版が持つ情報を管理するために version_of クラスを定義する．version_of クラスが保持する属性を**表 2** に示す．version_of クラスは，ある特定の時点での版の状態を保存するために用いられる．画像処理演算には数値データやヒストグラムのような画素値データ以外のものを結果として返すものも存在する．このため，画素値データだけでなく数値データや構造体も版として保存する必要がある．各種の型となる版の状態を保存するために，任意の型の値を保持できる Parameter クラスを作成し属性 this_version に用いている．また，root の version_of オブジェクトが参照する IDO は，GIO を作成するときに GIOFactory に対して指定された IDOS 側に存在する IDO である．このようにリモートのサイトに存在する IDO への参照は，IOS 内のメモリ表現のままでは保存することができない．この参照を Parameter クラスで保存するとき，CORBA オブジェクトに対して提供されている `object_to_string()` メソッドを用いてその参照を文字列に変換し保存する．

表 2 version_of クラスの属性
Table 2 Attributes of version_of class

型	属性名	内容
os_List<version_of* >	derived_from	この版の導出元の版のリスト
os_List<version_of* >	derive_to	この版から導出された版のリスト
char*	processing_name	この版を導出したときに用いた画像処理名
os_List<Parameter*>	used_parameters	画像処理に用いたパラメタ
char*	version_name	版の名前
CORBA::Boolean	stype	この版を保存するかどうかを示す条件
Parameter*	this_version	版として保存されるデータへの参照

表 3 GenericImage_i クラスの属性
Table 3 Attributes of GenericImage_i class

型	属性名	内容
version_of*	root	root の版への参照
version_of*	current	current の版への参照
os_Dictionary<char*,version_of*>	versions	版の名前と参照の対応
os_List<char*>	names_of_versions	全ての版の名前
int	version_num	版の総数
storage_type	stype	版の保存方法
char*	name	GIO の名前

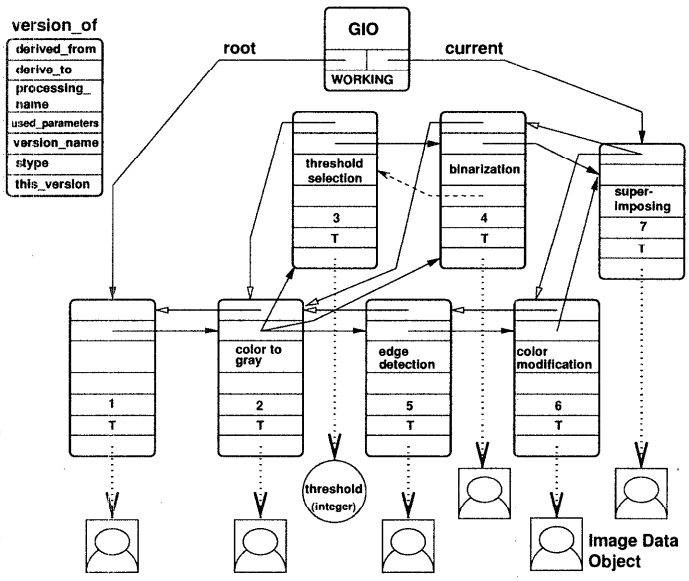


図 2 GIO の構造
Fig. 2 A structure of a GIO

さらに，version_of クラスは，それぞれの版を作成した手順に関する情報も管理する．version_of クラスがそれぞれの版の導出元（属性 derived_from），導出先（属性 derive_to）の版を示す属性を保持することにより，（derived_from relationship）及び融合関係（merged_from relationship）を表現する[11]．これらの属性によって，版の導出順序が表現される．また，画像処理に用いた画像処理オブジェクト名（属性 processing_name）及びそのパラメータ（属性 used_parameter）を属性として保持することにより，その版を導出したときに用いた画像処理に関する情報を保存できる．この実装ではある版が数値データや構造体を状態として保持しているとき，その版を画像処理の引数として用いることも可能である．この場合，属性 used_parameter には画像処理に用いた版への参照が挿入される．

version_of クラスによって図 2 に示すような版履歴の構造が表現される．しかし，この構造そのものを直

接操作することはプログラミングを複雑にする．そこで，GenericImage_i クラスを介して版の操作をアプリケーションプログラムに提供する．このクラスは，付録 A.3に示す GenericImage インタフェースを実装したものであり，このクラスの持つ属性を表 3 に示す．GenericImage_i クラスは，GenericImage インタフェースで定義されたオペレーションに対応するメソッドおよびクライアント側には公開せずに内部の実装だけで用いるメソッドを実装している．図 2 内の GIO が GenericImage_i クラスのインスタンスを表している．GIO によって version_of クラスによって表現された版履歴の構造をアプリケーションプログラムに対して隠蔽する．このために，version_of クラスに対する IDL によるインタフェースは定義せず，アプリケーションプログラムは GIO を介して間接的に版を操作する．

GenericImage_i クラスは，root の版（属性 root）と current の版（属性 current）を表す version_of オブジェクトへの参照を属性として保持する．それぞれ一番最初の版と現在操作している版を表している．さらに，GenericImage_i クラスは，version_of オブジェクトの名前とその version_of オブジェクトの参照の対応をリストとして保持する．これにより GIO が保持するすべての版をその名前から導くことができる．ただし，図 2 では簡略化のためこの属性は省略している．

4.2 版の生成および保存

画像処理を依頼するメッセージがアプリケーションプログラムから GIO へ送られたとき，GIO はさらに IPO へ実際の画像処理の依頼を行う．このように GIO によって IPO の分散は隠蔽され，アプリケーションプログラムは GIO が画像処理をメソッドとして保持するオブジェクトであるように扱うことができる．画像処理を実行した結果，新たな IDO が生成され，これが GIO の新たな版となる．このとき，GIO の内部では version_of オブジェクトを用いて版管理を行っているが，これらの情報は GIO によってカプセル化されるためアプリケーションプログラムは GIO の提供する版管理機構を利用することができる．すなわち，アプリケーションプログラムからは画像処理を実行したとき GIO の持つ画素値データが更新されたように見える．

図 3 に画像処理過程の例を示す．この例では，版データベースから IDO-A を版として保持する GIO を検索し，その GIO に対しさらにエッジ検出処理を実行する．画像処理は GIO を用いて以下の手順で実行される．

(1) アプリケーションプログラムが，画像処理のリクエストを imageProcessing() オペレーションを通して GIO へ送る．このときエッジ検出処理用の IPO の名前（Edge）と処理に用いるパラメータがオペレーションの引数として与えられる．

(2) GIO は，version_of オブジェクトが保持する IDO-A の参照を受け取る．

(3) GIO は，IDO-A の保持する画素値データをパラメータとして指定して IDOFactory に IDO-A のコピーの作成を依頼する．ただし，IDO-A が IDOS 内にある場合，後述する問題は起きないため IDOFactory にコピーの作成は依頼しない．

(4) IDOFactory は，IDO-A のコピーである IDO-A' を IPOS 内に作成する．

(5) IDOFactory は，IDO-A' への参照を戻り値として GIO へ返す．

(6) GIO は，指定された IPO の名前を用い Naming Service[14] を通して IPO を検索する．そして，GIO は IPOS 側の IPO に IDO-A' への参照を指定して画像処理の実行を依頼する．

(7) IPO は，画像処理を実行するために IDO-A' の保持する画素値データを受け取る．

(8) IPO は画像処理を実行し，その結果を IOS 側の GIO に返す．画像処理結果は画像処理アルゴリズム毎に画素値データや整数値等になり，型は異なったものとなる．このため，IPO が返す結果は IDL によって表現可能な任意の型の値を格納できる CORBA::Any クラスを用いる．特に，画素値データの場合は IDL で定義した sequence<octet> 型を用いて表現したバイト列を CORBA::Any クラスに挿入して転送する．

(9) GIO は画像処理結果から新たな版として IDO を生成し，その IDO が GIO の版であるという関

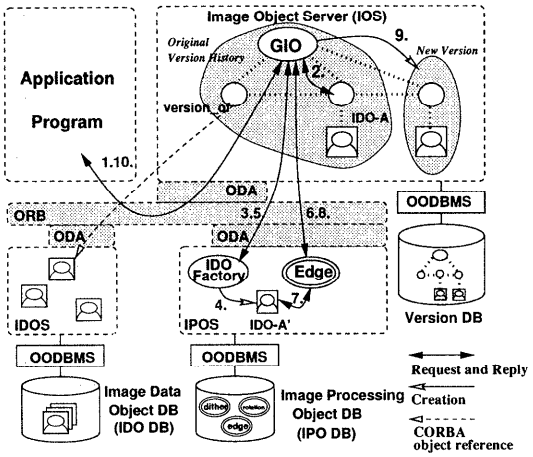


図 3 画像処理の実行
Fig. 3 Execution of an image processing

係を表現するために version_of オブジェクトの生成を行う．画像処理結果が数値データとして CORBA::Any クラスによって返された場合，CORBA::Any クラスが永続クラスでないため version_of オブジェクトは前述の Parameter クラスを用いてこの処理結果を保存する．アプリケーションプログラムから指定された画像処理のパラメータも保存され，version_of オブジェクトの導出元と導出先の関連も更新される．これらの情報が画像処理手順を再利用するときに用いられる．

(10) GIO は，アプリケーション側に新たに生成された版の名前を返す．

IPO に生成された画素値データは，以上の手順によって新たな版として版データベースに保存される．さらに画像処理が実行されたときも同様の手順が行われる．

この一連の手順の中で IDOFactory による IDO-A のコピーの作成は，以下の理由から行われる．もし，IDO-A の参照を直接 IPO に指定したとすると，IOS から IPO への画像処理の依頼と IPO から IDO-A への画素値データの取得の依頼がデッドロックを引き起こしてしまうからである．ただし，IDO が root の版の場合，IDO は IDOS に存在するためデッドロックの問題は生じない．したがって，この場合は IDOFactory によるコピーの作成は行わず，直接 IDO の参照を指定して画像処理を IPO に依頼する．

4.3 画像処理履歴の再利用

IPO DB に格納されている IPO は，単一の画像処理をメソッドとして保持しており，画像処理の依頼があったときその画像処理を実行しその結果を返す．一方，処理履歴の再利用を行うために，version_of オブジェクトによって管理された画像処理の導出手順の情報を属性として保持したものも IPO として実装できる．この IPO を特に履歴情報を持っていることから，画像処理履歴オブジェクト（History of Image Processing Object, HIPO）と呼び，IPO DB に格納されている IPO と区別する．

このように，一つの OMG IDL インタフェースに対して複数の実装を与えることが可能であるため，IPO のインタフェースに対して全く別の実装を与えることができる．IPO と HIPO は共通のインタフェースが定義されているため，GIO からはこれらを内部の違いを全く意識することなく同種のオブジェクトとして扱うことができる．

HIPO は，アプリケーションプログラムが GIO に対してオペレーション createIPO()（付録 A.3参照）を通じて IPO の生成を依頼したとき，その GIO が管理する版の導出履歴の属性に従って生成され版データベースに保存される．画像処理手順を再利用するとき，GIO が版として保存している途中の画像処理結果は不要である．このため，HIPO は処理手順に関する情報だけを保持する．ただし，その手順によってどのような結果が得られるかを示すために，その手順の処理前後の画像データのみをサンプルとして保存する．

HIPO に画像処理の依頼があったとき，HIPO は自分自身が保持する画像処理履歴に基づいて IPOS 側の IPO に画像処理を依頼し画像処理履歴を再現する．HIPO もまた画像処理履歴の情報に関して，その HIPO の元となった GIO と全く同じ構造の version_of オブジェクトによって履歴の表現を行っている．そして，HIPO がその画像処理手順を再現するとき，HIPO は各手順に対応した version_of オブジェクトが保持する画像処理名とパラメータを用いて IPOS 側の IPO に依頼する．

ある GIO が版として保持する IDO d に対し HIPO が保持する画像処理手順を復元するとき，その HIPO 内の処理手順の最後の版の version_of オブジェクトに対して以下に示す手続きを呼び出すことで実現される．

```
procedure imageProcessing(IDO d)
D : IPO に渡される IDO への参照のリスト
P : IPO に渡されるパラメタのリスト
begin
  if root の場合 then root の処理結果を d とする
  D := 空リスト; P := 空リスト;
  画像処理を依頼する IPO を属性 processing_name
  から決定する;
  foreach v in derived_from begin
    if 祖先の版 v が未処理 then
      v->imageProcessing(d);
    祖先の版 v の持つ IDO を D に追加;
  end
  foreach p in used_parameters
    if パラメタとして祖先の版を用いた then begin
      パラメタとして用いた祖先の版 v' を
      p から決定する;
      if 祖先の版 v' が未処理 then
        v'->imageProcessing(d);
      祖先の版 v' の保持する値を P に追加;
    end
    else
      p の保持する値を P に追加;
  D,P を引数として IPO に画像処理を依頼;
end.
```

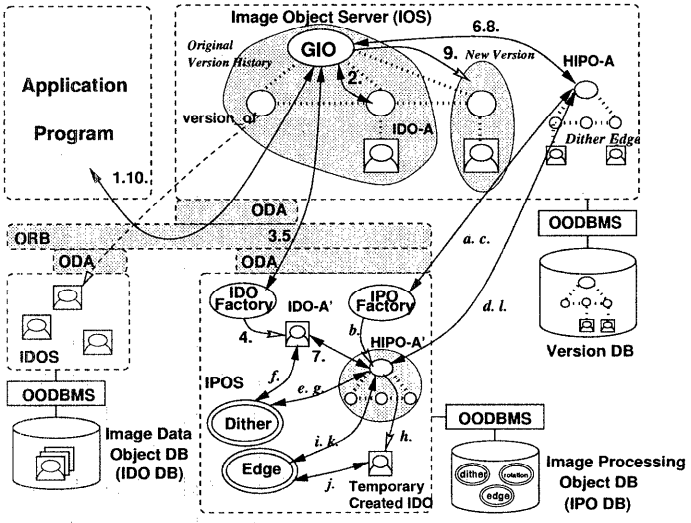


図4 画像処理履歴の再利用
Fig. 4 Reusing a history of image processing

先頭の版から順に画像処理を実行すると，図2の2番目の版のようにそこから複数の版を生成しているような場合や7番目の版のようにその版を生成するために複数の版を用いている場合，手順が分岐することが原因となり処理手順の復元は複雑となってしまう．そこで，再帰的手法を用いることにした．

この手続きを用いて画像処理手順を再利用するとき，その画像処理手順の途中で作成されるIDOは必要ない．IOS側にHIPOを置いたままで画像処理手順を再現すると，その手順を再現するときに生じる途中結果のIDOをIOSとIPOSの間で転送することになる．これは明らかにデータ転送コストの面から無駄な処理であり，効率が悪い．一般に画像処理手順のデータサイズは画素値データのサイズと比較すると非常に小さい．このため，画像処理手順の情報をIPOS側に転送しIPOS内で画像処理手順を再現することによって，途中結果のIDOの転送を省略する．

画像処理手順を再利用するときの例を図4に示す．この例は，ディザ処理，エッジ検出処理の順で実行された画像処理手順を保持するHIPO-Aに図3と同様のGIOに対して処理手順の再現を依頼したものである．図4中の1から10までの番号は，それぞれ図3の番号に対応している．GIOがHIPO-Aに画像処理を依頼したとき，HIPO-Aは自分自身のコピーHIPO-A'を作成することをIPOS側のIPOFactoryオブジェクトに依頼し（ステップaからc），HIPO-AはHIPO-A'に処理を依頼する（ステップd）．HIPO-A'は，IPOS内で画像処理手順を再現し（ステップeからk），その処理手順の最終的な結果のみをIOS側のHIPO-Aに返す（ステップl）．このHIPOのコピーによって行われる一連の処理は，GIOに対しては隠蔽されており，GIOは単にHIPO-Aが画像処理を実行したかのように見える．このため，GIOはHIPOとIPOを全く区別する必要はなく，HIPOをこのシステムに容易に組込むことが可能である．

4.4 実装システムの性能評価

この実験では，画像処理履歴に含まれるIPOの個数が2個から10個までのHIPOに対して画像処理の依頼を行ったときの処理時間を測定した．アプリケーションプログラム，画像オブジェクトサーバ，画像データオブジェクトサーバ，画像処理オブジェクトサーバそれぞれに対し各1台の計算機，合計4台の計算機を割り当てた．これらの計算機の構成を表4に示す．各計算機は100Mbpsイーサネットで接続されている．

表4 計算機構成
Table 4 Configuration of computers

	IOS, IDOS, IPOS	アプリケーション
計算機	Sun Ultra 30	Sun Ultra 1
CPU	UltraSPARC-II	UltraSPARC-I
クロック	248MHz	167MHz
メモリ	128MB	128MB
2次記憶	4.2GB	2.1GB
コンパイラ	Sun Workshop C++ 4.2	JDK 1.2
ORB	Orbix 2.1c	Java IDL
DBMS	ObjectStore R5.1	

この実験の結果を図5に示す．実験の条件として，履歴の長さを単純に比較するために，手順に含まれるIPOは全て回転処理を実行するIPOとした．また，画像処

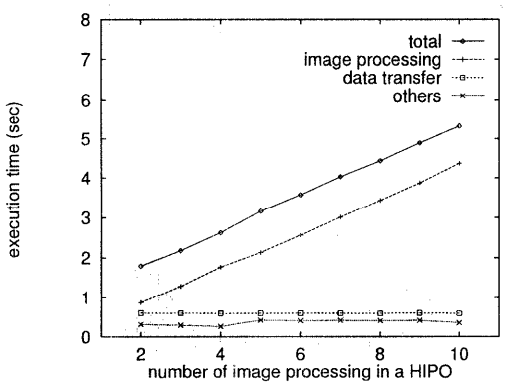


図5 画像処理履歴オブジェクトの実行時間
Fig. 5 Execution time of a HIPO

理を実行する画像データは，約128Kバイト（131243バイト）のものを用いた．

この実験から，全体の処理時間に対し画像処理と画像データの転送にかかる時間の占める割合は非常に大きいことが分かった．そこで，実験結果として次の三つの測定時間を示す．

- アプリケーションプログラムがHIPOの持つ処理履歴の実行を依頼してからアプリケーションプログラムに結果が戻って来るまでの時間（図4ステップ1～10）
- 画像処理にかかる時間（図4ステップe～k）
- 画像データの転送にかかる時間（図4ステップ3～5およびステップl）

その他に各プロセス間でのコネクションの確立およびリクエストの送信，IOS内での版の作成，HIPOのコピーの作成等が実行されているが，これらの処理時間は非常に短いため，図5中では一つにまとめて示している．図5が示すように，画像処理にかかる時間は処理の実行回数に比例して単調増加している．一方，画像データの転送は画像処理の実行回数とは無関係に2回行っているため，転送時間はほぼ一定となっている．また，その他の処理に占める時間もほぼ一定である．

したがって，処理全体にかかる時間はこれらの時間の合計であるため，画像処理の実行回数とは比例せず回数が多くなるほど効率が良くなっている．例えば，画像処理手順に10個のIPOが含まれるときの処理全体の実行時間は2個のときの約2.96倍となっている．すなわち，処理履歴の長さが増加したとき，全体の実行時間の増加に影響を及ぼすのは画像処理の実行回数だけであり，その他の処理は影響を及ぼさないためである．したがって，HIPOによって画像処理手順を再利用するとき，画像処理の依頼を逐次画像処理サーバに繰り返すときよりも効率良く行える．

4.5 アプリケーション例

図6に版管理機構を利用したアプリケーションの例を示す．この例は，医療画像データベースのアプリケーションプログラムであり，Javaを用いて作成されている．図6のメイン画面の左上の画像は，画像データベースから検索した胸部の断面のCT画像であり，一番最初の版に対応している．また，画面内の画像の下に表示されている番号が版の名前に対応している．この例では，左上の1番の画像から画像処理を平滑化，エッジ検出，色調変換，セグメント分割，重ね合わせの順に実行したことによって五つの版が新たに生成されている．特に，2番目の版は3番と5番の版を作成するために2回利用されている．

画像処理の実行は，アプリケーションプログラム内のプルダウンメニューから選択することによって対話的に行う．画像処理の要求はアプリケーションプログラムからIOSへと送られ，IOS側で画像処理が実行される．例えば，アプリケーションプログラムから平滑化処理をGIOに依頼する場合，平滑化処理のメニュー選択のイベントに次のようなコードを記述することによって処理の依頼を行う．

```
GenericImage gio;
...
// gioの参照の取得
...
// 画像処理引数の設定
parameter [] pars = new parameter[2];
pars[0] = new parameter();
pars[2] = new parameter();
Any val1 = orb.create_any();
Any val2 = orb.create_any();
val1.insert_long(window_size);
val2.insert_double(noise_variance);
pars[0].value(val1);
pars[0].value(val2);
// 融合関係として用いる版
String [] s = {};
try {
  gio.imageProcessing("AdaptiveSmoothing",
                      s, pars);
}catch(GenericImagePackage.InvalidArg e){
  ... // 例外処理
}
```

画像処理や版管理の機能はIOSによって全て提供されるため，このようなアプリケーションを作成する際にはGUIとIOSへの処理依頼に関するプログラミングの

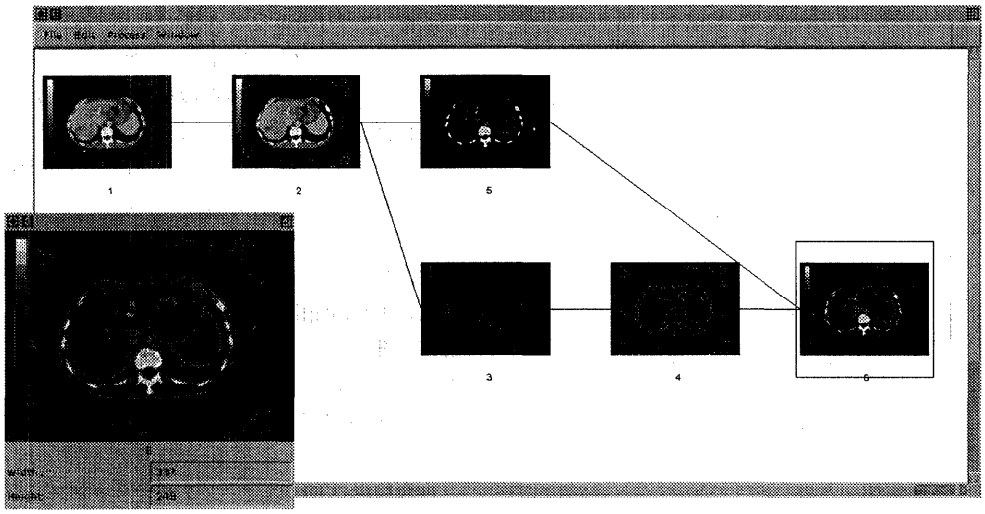


図6 版管理の例
Fig. 6 An example of version management

みとなる．したがって，アプリケーションプログラムの開発を非常に容易に行うことができる．

また，このアプリケーションで作成した画像処理手順はHIPOとしてその手順を再利用することができる．この場合，アプリケーションプログラムからIOSへHIPOの作成の依頼を行うことによって，図6の1～6の順序に画像処理を実行するHIPOが作成され，この手順を再利用することが可能になる．

画像処理の手順を作成・管理するという点で，このアプリケーションプログラムはAVS(Application Visualization System)のネットワークエディタを用いた画像処理のデータフロー作成に類似している[1]．しかし，AVSではユーザが開発した画像処理手順はユーザ自身がファイルに保存して管理する必要があるが，本研究のアプリケーションプログラムでは版管理機構が手順を版データベースに保存するためその必要がない．

5. むすび

本論文では，分散環境下において画像データベースのための版管理の機能を提供する版管理機構の実装について述べた．この版管理機構を実現することによって，ネットワーク上に分散した既存の画像データベースに対する版管理が提供され，アプリケーションプログラムが対話的に画像処理を実行したときに生じる画像処理結果と画像処理手順を再利用することが可能になる．

現在の実装の段階では，HIPOとして保存された画像処理履歴は名前を用いて検索する手段しか提供されていない．履歴内容に基づいて効率的に検索する方法について検討する必要がある．

謝辞 医療画像を提供して戴いた群馬大学医学部核医学講座の井上助教授に感謝致します．また，本機構の実装に関して金森研究室の赤石典久君の協力に感謝致します．本研究の一部は文部省科学研究費特定領域研究「高度データベース」（課題番号08244101），文部省科学研究費特別研究員奨励費（受付番号3473）による．

参考文献

1) Advanced Visual Systems, Inc.: *AVS User's Guide*, release 4 edition (1992).
2) Aritsugi, M., Tabata, M., Fukatsu, H., Kanamori, Y. and Funyu, Y.: Manipulation of Image Objects and Their Versions under CORBA Environment, *Proc. 8th Int. Workshop on Database and Expert Systems Applications (DEXA 97)*, pp. 86–91 (1997).
3) Dogac, A., Dengi, C., Kilic, E., Ozhan, G., Ozcan, F., Nural, S., Evrendilek, C., Halici, U., Arpinar, B., Koksal, P., Kesim, N. and Mancuhan, S.: METU Interoperable Database System, *SIGMOD Record*, Vol. 24, No. 3, pp. 56–61 (1995).
4) Forslund, D. W.: The Role of CORBA in Enabling Telemedicine, *Global Forum III : Telemedicin in Vienna(GLOB'97)* (1997).
5) 深津博樹, 有次正義, 金森吉成: CORBA環境下における画像処理オブジェクトの並列化, 第9回データ工学ワークショップ(DEWS'98) (1998).
6) 深津博樹, 田幡勝, 有次正義, 金森吉成: CORBA環境下におけるOODBMSの利用, 第55回情報処理学会全国大会論文集, Vol. 3, pp. 330–331 (1997).
7) IONA Technologies Ltd.: *Orbix 2 Program-*

ming Guide (1996).
8) Ishikawa, Y., Furudate, T. and Uemura, S.: A Wrapping Architecture for IR Systems to Mediate External Structured Document Sources, *Proc. of Fifth Int. Conf. on Database Systems for Advanced Applications(DASFAA'97)*, pp. 431–440 (1997).
9) Katoh, K., Morishima, A. and Kitagawa, H.: Navigator-based Query Processing in the World Wide Web Wrapper, *Proc. 5th Int. Conf. of Foundations of Data Organization(FODO'98)*, pp. 191–199 (1998).
10) Katz, R.: Toward a Unified Framework for Version Modeling in Engineering Databases, *ACM Computing Surveys*, Vol. 22, No. 4, pp. 375–408 (1990).
11) 川島亨, 田幡勝, 金森吉成, 増永良文: 画像オブジェクトの版管理モデル, 電子情報通信学会論文誌, Vol. J79-D-I, No. 10, pp. 843–852 (1996).
12) Koschel, A., Kramer, R., Nikolai, R., Hagg, W., Wiesel, J. and Jacobs, H.: A Federation Architecture for an Environmental Information System incorporating GIS, the World Wide Web, and CORBA, *Proc. Third Int. Conf./Workshop Integraing GIS and Environmental Modeling* (1996).
13) Object Design Inc.: *ObjectStore C++ API Reference Release 5.0* (1997).
14) Object Management Group, Inc.: *CORBAservices: Common Object Services Specification* (1997).
15) Object Management Group, Inc.: *The Common Object Request Broker: Architecture and Specification, Revision 2.2* (1998).
16) Orfali, R., Harkey, D. and Edwards, J.: *The Essential Distributed Objects Survival Guide*, John Wiley & Sons, Inc. (1996).
17) Pyarali, I., Harison, T. H. and Schmidt, D. C.: Design and Performance of an Object-Oriented Framework for High-Speed Electronic Medical Imaging, *Proc. of 2nd Conf. on Object-Oriented Technologies & Systems(COOTS)*, pp. 191–208 (1996).
18) Sun Microsystems, Inc.: *Java Platform 1.2 API Specification* (1998).
19) 田幡勝, 深津博樹, 有次正義, 金森吉成: 分散環境における画像オブジェクトの版管理の設計, 第8回データ工学ワークショップ (DEWS'97), pp. 179–184 (1997).
20) 谷崎正明, 嶋田茂: 地理空間オブジェクトトレーダー (GSOT) 構想と仮想都市空間構築への適用方式の検討, アドバンスト・データベース・シンポジウム'97(ADBS'97), pp. 41–50 (1997).
21) Tobar, C. M. and Ricarte, I. L. M.: Multiware Database: A Distributed Object Database System for Multimedia Support, *Open Distributed Processing: Experiences with Distributed Environments*, Chapman & Hall, pp. 439–450 (1995).

付録 OMG IDL によるインタフェース定義

A.1 interface ImageData

```
// 画像データのストリーム
typedef sequence <octet> bytes;
enum ImageType { // 画像フォーマットの定義
   COLOR_IMAGE,          // カラー画像
   GRAY_VALUE_IMAGE,     // 濃淡画像
   BINARY_IMAGE          // 2値画像
};
interface ImageData { //IDOのインタフェース
   bytes getData();      //画像データの取得
   ImageType getType();  //画像のフォーマット
   long getWidth();      //横サイズ
   long getHeight();     //縦サイズ
   short getColors();    //画像の階調数
   long getSize();       //画像データのサイズ
};
```

A.2 interface ImageProcessing

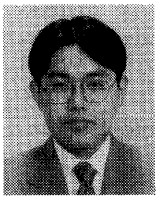
```
typedef sequence <ImageData> ImageDataList;
typedef sequence <any> anyList;
typedef sequence <TypeCode> TypeCodeList;
//IPOのインタフェース
interface ImageProcessing {
   exception preProc {string reason;};
   any executeProcessing( //画像処理の実行
      in ImageDataList images,
      in anyList arguments)
      raises(preProc);
   //画像処理に必要な画像データ数
   short numOfImages();
   //画像処理に必要な引数の情報
   TypeCodeList getArgInfo();
   ImageData sample(); //画像処理前のサンプル
   //画像処理後のサンプル
   ImageData processedSample();
   string manual();     //画像処理マニュアル
};
```

A.3 interface GenericImage

```
enum storage_type // 版を保存する条件[11]
   {transient,working,selective,complete};
// GIO のインタフェース
interface GenericImage :  ImageData {
   //例外の定義
   exception InvalidArg {string reason;};
   exception NotImageData {};
   exception NotValue {};
   exception AlreadyUsed {};
   exception DeletedRoot {};
   // 画像処理を実行
   string imageProcessing(
      in string processing_name,
      in stringList versions,
      in parameters arguments)
      raises(InvalidArg);
   // current の版の名前
   string nameOfCurrent();
   // current の版の全ての子孫の版の名前
   stringList getDescendants();
   // current の版の全ての祖先の版の名前
   stringList getAncestors();
   // current の版を作成した画像処理名
   string imageProcessingName();
   // 画像処理に用いた引数
   parameters imageProcessingParameters();
   // current の版の変更
   boolean gotoVersion(
      in string version_name)
      raises(InvalidArg);
   // 全ての版の名前のリストを取得
   stringList namesOfVersions();
   long numberOfVersions();   // 版の総数
   // current 以降の版の削除
   string deleteCurrent()
      raises(DeletedRoot);
   //版の保存に関するオペレーション
   void setStorageType(
      in storage_type type);
   storage_type getStorageType();
   void setCurrentStorageType(
      in boolean type);
   boolean getCurrentStorageType();
   //数値化した版に関するオペレーション
   TypeCode getVersionType();
   any getParameterizedData()
      raises(NotValue);
   //画像処理履歴から IPO を生成
   void createIPO(
      in string processing_name)
      raises(InvalidArg);
};
```

(平成10年12月20日受付)
(平成11年 3 月27日採録)

(担当編集委員 宝珍 輝尚)

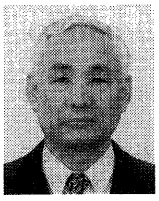


田幡　勝（学生会員）

1995 年群馬大・工・情報工卒．1997 年同大大学院博士前期課程（情報工学専攻）修了．現在，同大大学院博士後期課程（電子情報工学専攻）在学中．電子情報通信学会会員．

有次　正義（正会員）

1991 年九大・工・情報工卒．1996 年同大大学院博士後期課程了．1994–1995 文部省派遣留学生として McGill 大 (院) 在籍．博士 (工学)．1996 年から群馬大・工・情報工助手．並列分散データベース，マルチメディアデータベース等に興味を持つ．電子情報通信学会会員．



金森　吉成（正会員）

1969 年東北大大学院博士課程（電子工学専攻）修了．工学博士．東北大電気通信研究所助手，同助教授，仙台電波高専教授を経て，1990 年群馬大・工・情報工教授．オブジェクト指向データベース，マルチメディアデータベースに関する研究に従事．ACM，IEEE-CS，電子情報通信学会各会員．