データベースシステム 104−12
（1995. 7. 18）

画像オブジェクトの版管理モデル

川島 享 † 田幡 勝 † 金森吉成 † 増永良文 ††

† 群馬大学工学部情報工学科 †† 図書館情報大学

画像処理の適用により生成される画像オブジェクトを版として捉え、それらの履歴を管理するために汎画像オブジェクトを導入した。本論文では、このモデルを拡張して、複数の汎画像オブジェクトを部品として持つ複合汎画像オブジェクトを提案する。これにより複数個の画像オブジェクト同士の統合が容易になり、かつ大画像や3次元画像など長時間の処理を要する場合に、ユーザの都合により断続的に処理することができるようになる。さらに汎画像オブジェクトのためのデータベースを用意することにより、必要なオブジェクトを永続オブジェクトとして管理し、再利用することが可能となる。

Version Management Model for Image Objects

Susumu Kawashima † Masaru Tabata † Yoshinari Kanamori †
Yoshifumi Masunaga ††

† Department of Computer Science, Gunma University
†† University of Library and Information Science

We have introduced a model of generic image object(GIO) that manages versions of the objects created by applying image processing functions. This paper proposes a composite generic image object(CGIO), which has image objects as part by extending the model.

The CGIO makes easily to integrate many image objects into an object. It is necessary for the long computing time to execute the processings of very large or three dimensional images. In this case, user occasionally wants to stop the image processings based on his schedule.

However, the CGIO can suspend and start again the image processings. Furthermore, we describe to prepare the database for the GIO, and to reuse persistent GIO managed in the database.

1 はじめに

これまで著者等は画像データベースシステムにおけるアプリケーション開発を容易に行なえる環境について研究してきた。画像データベースの特徴の1つである画像処理に焦点を絞り、これと画像データとをカプセル化し画像オブジェクト (Image Object 、以下IO) を構築することについて議論してきた [1, 2, 3]。さらに、このオブジェクトを異機種分散環境での利用に対応できるようにするため、ユーザには仮想的に画像データと画像処理をカプセル化したオブジェクトに見えるが、物理的には画像データと画像処理を分散的に配置し、必要に応じて動的に結合する画像オブジェクトモデル、およびそれのアーキテクチャを提案した [3]。

このアーキテクチャは、アプリケーション層、中間層、データベース層の3階層から構成されている。中間層の1つである画像オブジェクトサーバでは画像処理メッセージを実行する毎に、新たな画像オブジェクトが生成される。この生成された画像オブジェクトを版として捉え、その版の履歴を管理するための汎画像オブジェクトを導入するモデルについて議論した [3]。

本論文では、このモデルをさらに拡張し、複数の汎画像オブジェクトを部品として持つ複合汎画像オブジェクトについて述べる。この複合汎画像オブジェクトの導入により、複数個の画像オブジェクト同士の統合が容易になり、かつ大画像やCTスキャンの3次元画像など長時間に渡って画像処理を必要とする場合に、ユーザの都合により処理を断続することができることを示す。さらに、汎画像オブジェクトのためのデータベースを用意することにより、必要なオブジェクトを永続オブジェクトとして管理し、再利用することについて検討する。

2 画像オブジェクトの版管理モデル

従来、主にCAD分野においてデータベース内のインスタンス、およびスキーマ (クラス) の版管理について研究が行なわれている [6, 7, 8]。これらの研究における版管理では、個々の版の間の意味的関係を導出関係 (derived_from 関係) と version_of 関係 (kind_of 関係、IS_A 関係)

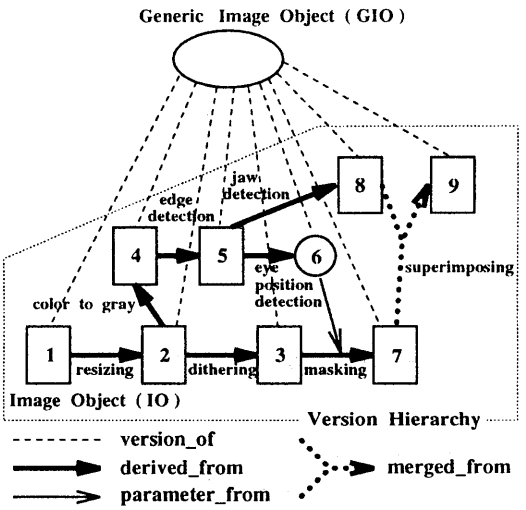

図 1: 画像オブジェクトの版管理モデル

で表現し、版階層構造 (version hierarchy) を形成する。そして、抽象化された "genric object" がこれらの版の共通の属性と個々の版の履歴構造を保持することにより、階層構造全体を統合的に管理するようになっている。

このような版管理手法を画像処理問題に適用することを考える。

2.1 画像オブジェクトの履歴管理

画像処理を実行する毎に画像データは変化していく。一般的に画像処理手法は種類も多く [9]、それ故、手法を選択する際に試行錯誤を繰り返し行なわなければならないこともある。その場合、ユーザは一処理前の画像から処理をやり直せた方が便利である。また、同じ画像を何度も利用して画像処理を行なうこともある。これらの処理過程を考えると、画像データの履歴を管理することの必要性は明らかである。

画像オブジェクトに画像処理を実行する毎に新たにIO が生成される。これらのIO をシステム側では版として見ることができる。一方、アプリケーション側からは画像オブジェクトのデータが変更されたように見える。そこで、これら両者の見方を統合するために、ユーザから見える画像オブジェクトを汎画像オブジェクト (Generic Image Object 、以下GIO) と見なすことにより、版管理問題として捉えるモデルを

提案した [3]。これらの例を示したのが図 1である。

図中の番号を付けた長方形 (画像データを保持) と円 (数値データを保持) が IO を表している。また、IO に対して画像処理を行なうと、結果として IO が生成されるが、その画像処理を矢印が表している。さらに、矢印の種類の違いが画像処理前後の IO の関係の違いを区別している。ほとんどの処理は、IO から IO が生成されるという意味で derived_from 関係であるが、その IO が画像処理のパラメータとして利用される parameter_from 関係、同一の GIO で管理されている複数の IO を融合した merged_from 関係も定義した [3]。さらに、GIO と個々の IO とを version_of 関係で関連づける。

図中では、楕円が GIO に当たり、点線が version_of 関係を示している。この関係は、GIO の版が IO であることを示し、IO はある時点での画像オブジェクトとなる。また、GIO は版の管理、操作を行ない、ユーザからの画像処理メッセージを IO に伝達し画像処理を命令する機能を持っている。一連の画像処理により、図 1のようなグラフ構造が生成される。これを版階層構造と呼ぶ。

このモデルではユーザが扱う画像オブジェクトは GIO であり、その GIO は意味的には 1 つの IO に順次処理を加えていくことにより、生成される画像オブジェクトから構成される。従って、GIO の管理する原画像 (つまり、画像処理を実行していない画像) は 1 つであり、複数の原画像から構成される画像 (例えば、大画像や 3 次元画像) をこのモデルでは扱うことはできない。次節では、このような複数の画像に対しても対応できるようにモデルを拡張する。

2.2 複合汎画像オブジェクトの導入

GIO の原画像が複数ある例を次に挙げる。

"3 種類 (地図、ひまわりの気象画像、人口密度図) の日本付近を示した画像があり、これらから関東地方を抜き出し、地図と気象画像、地図と人口密度図を重ね合わせて表示する" という場合を考える。

この手順としては、

(1) 3 種類の画像を用意する (DB から検索).

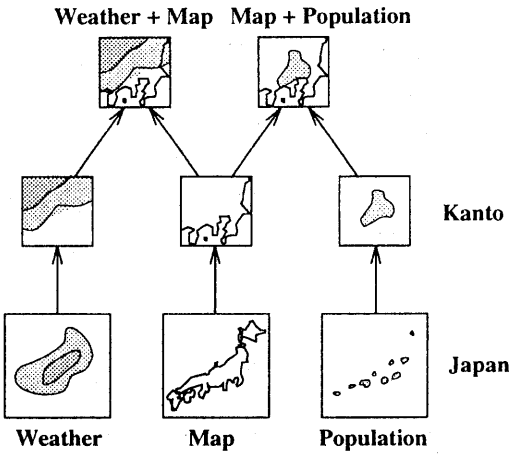

図 2: 複数の画像の重ね合わせ手順

(2) それぞれの画像から関東地方を抜き出す.

(3) 関東地方の地図と気象画像、および地図と人口密度図を重ね合わせて表示する.

であり、この様子を図 2に示す。

ここでは、原画像として 3 種類 (地図、気象画像、人口密度図) の画像が存在している。3 つの原画像を 1 つの GIO で扱うと、ユーザから見える 1 つの画像オブジェクトが異なる 3 つの画像オブジェクトを表現していることになり意味的に矛盾する。

従って、それぞれの原画像は 1 つの GIO で管理しなければならない。それ故、異なる GIO が管理している複数の IO を組み合わせて処理が行なえるモデルを考える必要がある。

本研究では、これら複数の GIO を扱うために、複合汎画像オブジェクト (Composite GIO、以下 CGIO) を導入する。この CGIO では、複数の GIO とを part_of 関係で関連づけることにより、GIO を統合的に管理することができる。

また、CGIO には GIO 同士を組み合わせて処理を行なうメソッドを導入する。メッセージに関しても CGIO から GIO、GIO から IO という順番で伝達される機能を持たせる。CGIO を用いて、例は図 3のように表される。ここでは 3 種類の画像 (地図、気象画像、人口密度図) を扱うので、これらを管理するために 3 つの GIO を必要とする。さらにこれらを統合して扱うために CGIO を設け、GIO と part_of 関係で関連づけている。画像処理に関しては、それ

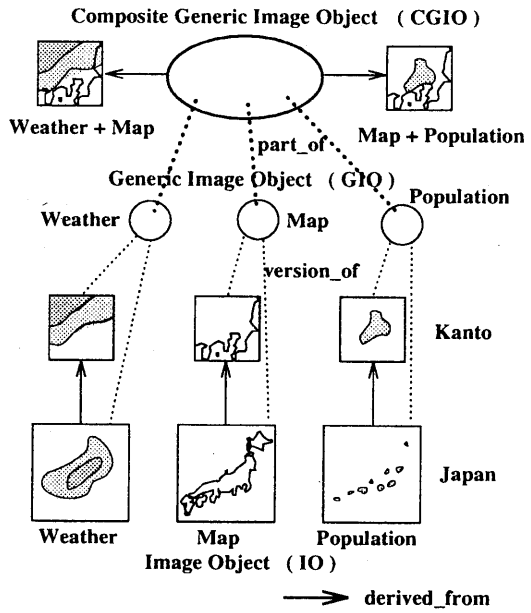

図 3: 複合汎画像オブジェクト

ぞれの日本付近の画像から関東地方を抜き出す処理を GIO に行なわせる。そこから、地図と気象画像、地図と人口密度図の重ね合わせを行なうために、個々に、CGIO にメッセージを送り、重ね合わせた画像を得る。

実際に画像処理を行なうのは GIO に管理されている IO であることに注意しなければならない。従って、CGIO の処理結果、すなわち複数の GIO を組み合わせた処理により生じる画像に対しては画像処理を直接は行なえない。この理由は、結果の画像を CGIO の版として管理しないからである。そこで、ここでは統合した画像にさらに処理を加えたければ GIO の管理下にある IO に処理を加えることになる。CGIO の処理は個々の GIO を組み合わせる処理だけに留める。

上述のようにして CGIO を導入すると複数の異なる画像を組み合わせなければならない処理も実行可能となった。ここで、GIO 相互の関係について考えてみる。上記のような複数の画像の重ね合わせでも、一定の法則がある場合とそうでない場合が考えられる。図 3の例は後者であり、次に前者の例を挙げる。

例えば、大画像を複数の小画像に分割して扱う場合、それぞれの GIO の関係には大画像上で

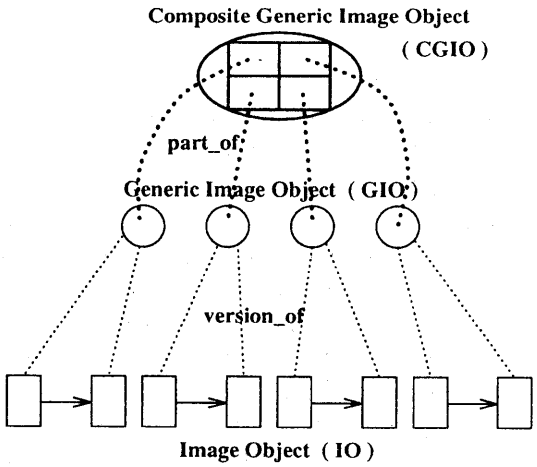

図 4: 大画像の履歴管理

の位置関係が反映される。また、CT スキャンの 3 次元画像に関しては複数の断面画像を再構築して立体に見せるが、この場合、GIO 相互の順序関係が重要となる。従って、GIO 相互の関係には次の 3 つが考えられ、その違いにより CGIO の機能を変える必要がある。

- GIO 相互の関係が一定の法則を持たない.
- GIO 相互の位置関係が重要である.
- GIO 相互の順序関係が重要である.

次に、大画像についての CGIO 、3 次元画像についての CGIO を述べる。

2.3 大画像の履歴管理

大画像 (例えば 10000 × 10000 画素) に関して画像処理を実行する場合、1 枚の画像を複数枚の画像に分割して長時間に渡って処理をする場合が考えられる。これを前節で述べた CGIO を用いて表現したものが図 4である。

分割した複数の画像をそれぞれの GIO で管理し、これらと CGIO を part_of 関係で関連づけて、GIO 相互の位置関係を管理することで大画像を 1 つのオブジェクトとして扱うことができる。複数の画像を組み合わせて処理を可能とするためにこれらに関するメソッドを CGIO で保持する。前述のようにユーザからのメッセージは、CGIO を通して個々の GIO に伝わり、GIO からそれぞれの IO に伝わる。また、大画像に対する直接の画像処理は行なわず、GIO を通して IO を画像処理する。

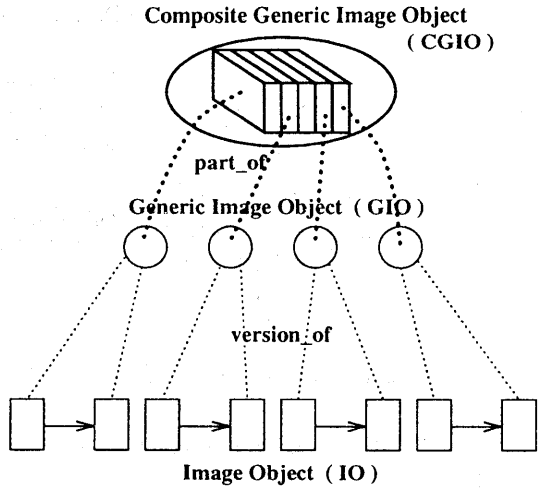

図 5: 3 次元画像の履歴管理

このように大画像を複数の小画像に分割し管理することにより、処理を小画像毎に個別的に行なうことができるので、長時間に渡る処理をユーザの都合により中断したり、再開したりすることができる。従って、ユーザにとっては処理に関する時間的柔軟性が増す。

2.4 3 次元画像の履歴管理

CT スキャンの 3 次元画像の場合を例にすると、複数の断面画像を再構成処理して、3 次元画像が構成されている。これらの多数の断面画像処理に長時間要する場合は、ユーザの都合により中断、再開することが容易にできる必要がある。このために個々の断面画像に GIO をそれぞれ割り当てて処理などの管理を行なわせ、それらと CGIO を part_of 関係で関連づけることにより 3 次元画像を 1 つのオブジェクトとして管理できるようにする。その様子を図 5に示す。また、GIO から 3 次元画像を構築するためのメソッドを CGIO で保持する。

前述の大画像の CGIO では GIO 相互の位置関係が重要であったが、3 次元画像の CGIO では GIO 相互の順序関係が重要となる。すなわち、順序関係が正しくないと正確な 3 次元画像が構成されない。

また、前述の場合と同様に構成された 3 次元画像に対する直接の画像処理は行なわず、GIO を通して IO に処理を適用する。CGIO にはメソッドとして、レンダリング処理や回転、移動処理などが用意されるのみである。

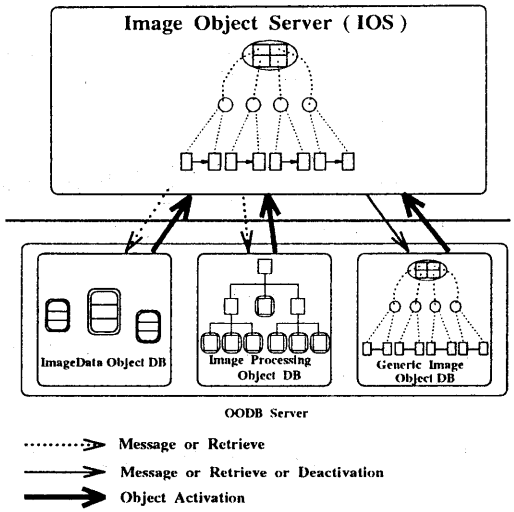

図 6: GIO DB を含んだアーキテクチャの拡張

3 画像オブジェクトのデータベースへの保存

これまで提案してきたモデル [3] では、IO をどのように構築するか、画像処理をどのように IO と動的に結合させるか、また、画像処理に伴い IO の履歴管理をどのように行なうかという視点から検討してきた。

このモデルでは、画像オブジェクトはアプリケーションが動作している時に有効で、終了と共にオブジェクトも削除されるものとしてきた。すなわち、ある処理を実行した画像オブジェクトを非永続オブジェクトと考えていた。

しかし、前節で述べたような CT スキャンの 3 次元画像、大画像などの場合にはユーザの都合により、数日、あるいは 1 週間ぐらいの長期に渡って処理を断続的に行なうことが考えられる。また、アプリケーションによっては処理結果を残して再利用したい場合もある。

このような場合には前回提案したアーキテクチャでは対応できない。そこで、IO、GIO、CGIO を GIO DB と名付けるデータベースに格納し、永続オブジェクトとして管理できるようにモデルを拡張する。その様子を図 6に示す。このようにしてアプリケーションを起動したときに GIO DB から格納したオブジェクトを活

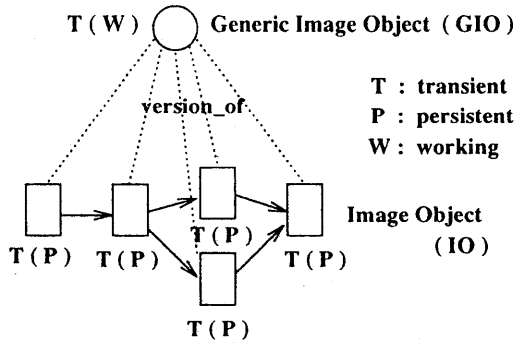

図 7: 一時的なオブジェクトと処理途中のオブジェクト (括弧内) の状態

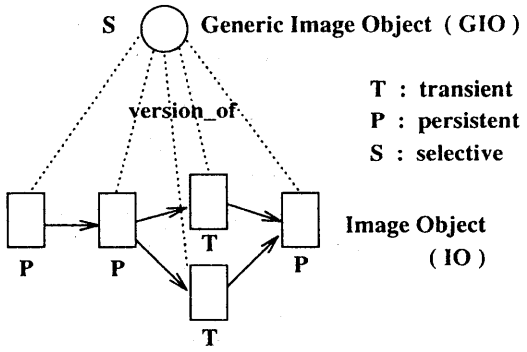

図 8: 必要なオブジェクトを選択した状態

性化することにより、再利用することができる。

3.1 画像オブジェクトの保存方法

アプリケーション実行時に動作している IO、GIO、CGIO は画像オブジェクトサーバ (Image Object Server、以下 IOS) で管理されている。GIO DB を設けることによりアプリケーション終了時に IOS 上にあるオブジェクトの中から必要とするオブジェクトを GIO DB に格納することができる。ここでユーザがオブジェクトの必要性を判断し、それを識別できる機能が重要となる。オブジェクトの保存方法としてアプリケーションの種類により、次の 4 つが考えられる。

- 保存しない.
- 一連の画像処理の途中であるから関連するオブジェクトをすべて保存する.
- 一応画像処理は終了したが、また再開する可能性があるので必要度の高いオブジェクトを保存する.
- 処理の最終結果だけを保存する.

上述のようにオブジェクトの保存方法の違いを区別するために、オブジェクトに保存状態を記述する属性を与える必要が生じる。GIO、CGIO の保存方法は上述の 4 種類が考えられるが、一方、IO はデータベースに保存するか否かの 2 種類しかない。そこで、IO の状態を次の 2 つの状態で表す。

- 一時オブジェクト (**transient**)
- 永続オブジェクト (**persistent**)

また、GIO、CGIO の状態は次に示す 4 種類である。

- 一時オブジェクト (**transient**)
- 関連するすべてのオブジェクトを保存 (**working**)
- 必要なオブジェクトのみ保存 (**selective**)
- 最終結果のみを保存 (**complete**)

GIO は 4 つの状態に応じて、管理している IO の状態を変更する。CGIO の場合も同様であり、4 つの状態に応じて CGIO の状態が GIO に伝わり、その状態の変更により IO の状態が決定される。

アプリケーションが終了する時、システムが上述したような状態を見て、各オブジェクトをデータベースに保存するか削除する。

次に GIO と IO の状態の様子を例を用いて説明する。

(1) まず、オブジェクトを作成した時には、必ず一時オブジェクト (transient) の状態になる。この状態を変更せずにアプリケーションを終了させるとすべてのオブジェクトは削除される。これを図 7に示す。同様に、まだ処理途中であるからすべてを保存したい場合、GIO を working 状態に変更すると IO すべてが persistent 状態になる。これは、図 7の括弧内で示された状態である。

(2) 次に、一応処理を完了したが、再び処理を再開するかもしれない場合は、必要なオブジェクトだけを保存する。その例を図 8に示す。この場合、GIO の状態を selective に変更すると、ユーザが任意の IO を保存することができるが、ここでは版階層構造の開始点、終了点、分岐点、

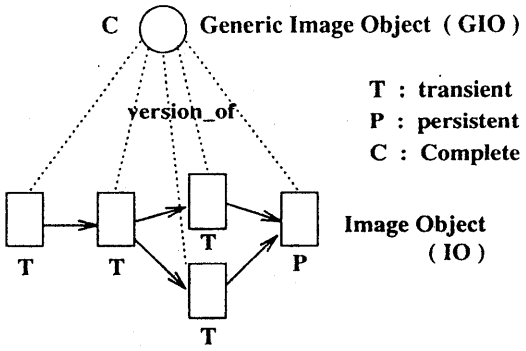

図 9: 処理を完了したオブジェクトの状態

融合点の IO のみを保存するように persistent 状態に変更している。ユーザはこれらの IO の版以外は、一時的であり、必要としないと判断している。

(3) 最後に、大部分のアプリケーションにとっては画像処理の最終結果のみあればよいものと考えられる。この時、GIO の状態を complete に変更することにより、管理している IO の最後だけが persistent 状態に変更され、図 9に示すように最終結果だけを保存する。

3.2 GIO DB の特徴

GIO DB の特徴を述べる前に図 6に示された他の 2 つのデータベースについて簡単に述べる。まず、画像データオブジェクト (ImageData Object、以下 IDO) DB は、IDO を格納するオブジェクトである。IDO は画像処理を行なう前の原画像に相当する。このオブジェクトは画像処理に関するメソッドを保持せず、画像データのみを持っている。

また、画像処理オブジェクト (Image Processing Object、以下 IPO) DB は個々の画像処理を 1 つのオブジェクトとして捉え、このオブジェクトを格納している。IOS 上で IO が画像処理を実行する場合、その IPO が検索され、IO と動的に結合された形で画像処理を実行する。

GIO DB は、IOS 上で動作している IO、GIO、CGIO を前述の 4 つの状態を反映した永続的なオブジェクトにするためのデータベースである。このデータベースの特徴を次に挙げる。

- ユーザに直接見えるオブジェクト (GIO や CGIO) とユーザには直接見えないオブジェクト (IO や GIO) を識別して格納する.
- 同じ IO が複数格納されるのを防ぐために、複数の GIO と IO が version_of 関係にある GIO をクラスタリングして格納することが可能である (複数の CGIO と part_of 関係にある GIO のために CGIO をクラスタリングして格納することも可能).
- 保存するオブジェクトはアプリケーションに依存するので基本的にユーザに管理される。すべてのユーザの共有資源とするためにはアプリケーションに依存して進化していくデータベースを構築する必要がある。現在、この問題については検討中である.

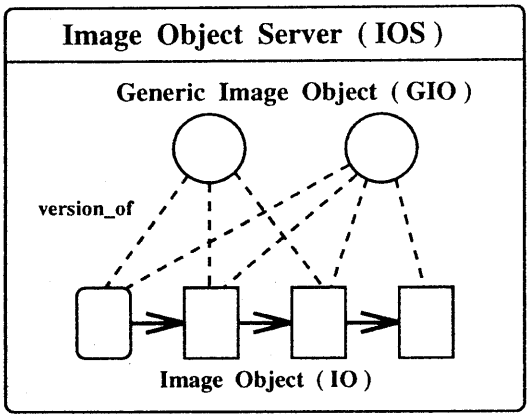

図 10: IOS 上の GIO と IO

ここでは 2 番目の特徴について詳しく言及する。IOS 上では図 10に示すように複数の GIO が 1 つの IO と version_of 関係で関連づけることを許している。これは IOS 上にすでに必要となる IO が存在するならば、資源、時間共に節約するためにそれを利用するからである。

これらの GIO を GIO DB に格納する場合、別々に GIO を格納すると、それに伴い IO もそれぞれの GIO の下に格納されてしまうので、同じ IO が重複して格納されることになる。これを避けるために、GIO をクラスタリングして格納する仕組みを提供する。

上述のような特徴を持った GIO DB を導入し、IO、GIO、CGIO の各オブジェクトに状態の属性値を与えることにより、アプリケーション開発を支援する再利用可能なオブジェクト構築が可能となる。

次節では、このようなオブジェクトの利用法について考えてみる。

4 IOSの利用法

IOSは、アプリケーションを起動する時に利用され、種々のオブジェクトを生成する。アプリケーション開発をするためにはデータベース、プログラミング言語、GUI、画像処理に関する知識が必須である。これらをすべて習得することはユーザにとって負担が大きい。そこで、著者等はこれらを統合したスクリプト言語(画像DBアプリケーション開発言語)の研究も行なっている[4, 5]。

この言語の特徴は、画面(ウィンドウ)、ボタン、画像などをすべてオブジェクトとして捉え、処理の実行などをメッセージとして送るオブジェクト指向的なスクリプトになっていることである。このスクリプトの中から画像や画像処理に関する部分が図6に示されるIOSにメッセージとして渡されることによりGIOやCGIOが構築される。

しかし、ユーザのスクリプトの記述法はさまざまである。記述によっては冗長なIOが生成される可能性がある。そこで、スクリプトの構文解析をする際には、スクリプトから版階層構造を生成し、この構造を見ることにより一度構成したIOを再利用するという最適化が重要となる[5]。このようにして時間がかかる画像処理を効率化してアプリケーションを高速に動かすことができる。

また、画像処理をスクリプトで記述することが不可能な場合、すなわちユーザが画像データを見ながら対話的に処理を行なわなければならないことも想定している。対話処理では、画像処理履歴などを表示できる対話ツールを用意し、ここからユーザの指示をメッセージ形式でIOSに渡すことでGIOやCGIOを構築する。

5 おわりに

前回提案した汎画像オブジェクトのモデルを拡張して、複数の汎画像オブジェクトを部品として持つ複合汎画像オブジェクトについて述べた。これにより複数個の画像オブジェクト同士の統合が容易になり、かつ大画像や3次元画像など長時間の処理を要する場合に、ユーザの都合により断続的に処理することができるようになった。さらに汎画像オブジェクトのためのデータベースについても検討した。

謝辞

本論文は95年1月の本研究会における研究発表に対しての大阪大学の下條助教授の御助言が動機となった。ここに深く感謝致します。また、日頃討論頂く、仙台電波高専の脇山氏に感謝致します。

参考文献

[1] 脇山俊一郎, 大津浩二, 福田紀彦, 金森吉成, 増永良文. オブジェクト指向データベースシステムにおける画像オブジェクトの構成と実装. 情報処理学会データベースシステム研究会, 94-22, 7月 1993.

[2] 脇山俊一郎, 川島享, 金森吉成, 増永良文. 分散環境における汎用画像オブジェクトの構成. 情報処理学会データベースシステム研究会, 99-11, 7月 1994.

[3] 川島享, 金森吉成, 増永良文, 脇山俊一郎. 画像オブジェクトサーバにおける版管理モデル. 情報処理学会データベースシステム研究会, 101-17, 1月 1995.

[4] 大津浩二, 松田宜之, 金森吉成, 増永良文, 脇山俊一郎. 画像DB用アプリケーション開発言語の設計. 情報処理学会データベースシステム研究会, 100-7, 10月 1994.

[5] 松田宜之, 大津浩二, 金森吉成, 増永良文, 脇山俊一郎. 画像DBアプリケーション開発言語の実装. 情報処理学会データベース研究会, 102-3, 3月 1995.

[6] Randy H. Katz. Toward a Unified Framework for Version Modeling in Engineering Databases. *ACM Computing Surveys*, Vol. 22, No. 4, December 1990.

[7] G. Talens, C. Oussalah, and M.F. Colinas. Versions of Simple and Composite Objects. *Proceedings of the 19th VLDB Conference*, pp. 375–408, 1993. Dublin,Ireland.

[8] Won Kim. *Introduction to Object-Oriented Databases*. The MIT Press, 1990.

[9] 画像処理サブルーチン・パッケージ SPIDER USER'S MANUAL. 電子技術総合研究所, 1980.