システムソフトウェアと
オペレーティング・システム 81－1
（１９９９．５．６）

多態的表現を可能にする永続オブジェクト管理機構

高野了成†，佐藤元信†，早川栄一‡，並木美太郎†，高橋延匡‡
†東京農工大学　工学部　‡拓殖大学　工学部

〒184-8588　東京都小金井市中町 2-24-16
E-mail：takano@os-omicron.org

本稿では，多態的表現を可能にする永続オブジェクト管理機構の設計と実現について述べる．計算機を利用して発想活動の支援を行う場合，データの性質に応じた処理の実行や，データ間の関連を効率的に扱うことが要求される．このような要求に対して筆者は「意紙」というデータモデルを提案している．マルチメディアデータなど多態的な性質を持つデータは，一つの実体に対して認識処理やモデル化の違いから複数の表現形式を持つことができる．そこで，関数表を利用した多態的なプログラミングインタフェースを提供し，要求に対してより効率的な表現形式を選択するための機構を提供する．また，データ間の関連をポインタで記述できるようにし，ポインタを含んだデータ構造に対する永続化操作を容易にする．

A Persistent Object Management Machanism with Polymorphic Representation

TAKANO Ryousei †, SATO Motonobu †, HAYAKAWA Eiichi ‡,
NAMIKI Mitarou † and TAKAHASHI Nobumasa ‡
† Department of Computer Science, Tokyo University of Agriculture and Technology
‡ Department of Computer Science, Takushoku University

2-24-26 Nakacho Koganei, Tokyo, 184-8588 Japan
E-mail：takano@os-omicron.org

This paper describes a design and implementation of a persistent object management machanism with polymorphic representation. A system supporting creative works requires to apply operations with characteristic of data, and manage relations of data efficiently. We have proposed a data representation model called" ISHI" for these demand. Multi-media object includes multiple representation in a object that produces from pattern recognition or data abstraction. " ISHI" system supports the selection to optimize data types for more efficient operations with polymorphic programming interface using function table. And also, programmers can describe relations of data by address. Therefore, it is able to persist simple data structure that contains data link using pointer notation.

1. はじめに

計算機の普及により，個人で計算機を所有し，文章作成や研究開発などの発想活動に利用されるようになってきた．このような計算機環境において，個人の知識を計算機上で効率的に蓄積，編集，再利用することが重要であり，データ管理機構は対象となるデータの性質に応じた抽象化を提供する必要がある．しかし，従来のファイルモデルでは，一つの実体が複数の表現方法を持つデータを多態的に扱うことや，それらのデータ間における密接な関連性を管理することは困難であった．そこで，データ管理機構には，多態的表現を実現できるデータモデルを提供すること，データ間の関連付け処理に対する記述を容易にし，高速に処理することが要求される．

筆者は，プログラマに対して，このような要求を実現することを目的としたデータモデルである「意紙」[1] を提案し，実装してきた．

2.「意紙」の概要

2.1「意紙」の目的

検索や編集などの操作は，データ型ごとに操作に対する適応性が異なる．例えば，一般的に検索処理はテキストに対して行う方が効率的であり，全体を俯瞰したい場合は，図形の縮小表示が便利である．「意紙」は一つの実体に対する複数の表現方法を包含し，処理内容ごとに，より効率的な処理に適したデータへの変換機構を提供する．このように多態性を利用した資源管理を行うことで，マルチメディアデータなど多態性を持つデータを効率的に処理することを目的とする．

また，データ間の関連を多く含んだ永続的なデータに対するアクセスを容易にするために，プログラマに対して記憶階層を意識しない永続化機構を提供することを目的とする．

2.2「意紙」の設計方針

(1) オブジェクトとしての抽象化

データに対する操作として，マルチメディアデータのようにリアルタイム性が要求されるものなど，データの性質に応じて適切な処理を提供できることが要求される．そこでデータの性質を表すデータ型と手続きを組で扱えるようにデータをオブジェクトに抽象化する．

(2) システム共通のデータ型管理

あるアプリケーション用に作成した手続きを，他のアプリケーションでも再利用可能にするには，オブジェクト間のインタフェースを統一する必要がある．そして，システムで統一的にデータ型を管理して，動的にデータ型を追加，削除できるようにする．

(3) ワンレベルストアの提供

単一アドレス空間では，空間切換えのオーバヘッドなしにデータとアドレスを一意に対応させることができる．つまり，オブジェクトをアドレスで識別することが可能になり，オブジェクト間の関連性をアドレスで記述できる．したがって，言語のポインタで関連性を容易に記述することができる．このオブジェクトとアドレスの対応を永続的に保証できるようにワンレベルストアを提供する．

2.3「意紙」の特徴

(1) 多態的なデータを効率的に扱うことができる

マルチメディアデータなど多種多様な性質を持つデータは，一つの実体に対して認識処理やモデル化の違いから複数の表現形式を持つことができる．このようなデータの集合に対して，関数表を利用した多態的なプログラミングインタフェースを提供し，要求された処理に対して，より効率的な表現形式を選択するための機構を提供する．

(2) データ間の関連性に対する整合を保証する

発想活動で対象となるデータ間の関係は非常に密接であり，個々の関連が思考の過程など，重要な意味を持つ場合もある．したがって，関連性の整合を保証することは重要である．そこで，関連を管理し，関連に対する変更が発生した場合，データを参照しているアプリケーションに対して変更を通知する機構を提供することで，関連の一貫性を保証する．

(3) 記憶階層を意識しない永続性を提供する

ファイルによる永続化は，言語処理系におけるデータ構造を明示的に直列化する必要があるなど，言語と永続記憶におけるデータ表現のギャップが問題になる．また，利用開始，終了を明示的に記述する必要がある．そこで，二次記憶へのアクセスをアプリケーションから隠蔽し，ポインタを含んだ複雑なデータ構造に対する永続記憶の操作を容易にする．

2.4 永続オブジェクトと「意紙」

(1) 永続オブジェクト

永続記憶は，タスクの生成，消滅に無関係に永続的に利用できるワンレベルストアな記憶領域である．永続記憶は，データ格納用の永続データと，手

続き格納用のモジュールに分類できる．永続データの性質を表すデータ型を属性と呼び，永続データは必ず属性を持つ．また，モジュールは手続きの集合のコンパイル単位である．永続オブジェクトは永続データとその属性に対するモジュールから構成される．

(2)「意紙」

「意紙」は，オブジェクトの多態的な表現を可能にするためのデータモデルであり，永続オブジェクトの集合として表現される．また，「意紙」は必須ではないが名前を持つこともできる．名前を持つ「意紙」は，名前を利用して，明示的にマップ要求を指示することができるので，プログラムからのエントリポイントとして利用できる．

2.5 「意紙」におけるプログラミングモデル

2.5.1 クラス

属性はクラスとして記述し，クラスによって永続オブジェクトの形式を定義する．「意紙」は永続オブジェクトのインスタンスの集合として表す（図1）．通常のオブジェクト指向言語は継承を提供しているが，例えば，手書きストロークと文字コードを一つのオブジェクトとして表現するとき，手書きストロークと文字コードの間に関連はあるが，継承関係で記述することはできない．そこで，「意紙」に含まれる永続オブジェクトの属性間には継承関係が存在せず，言語の継承関係と独立した属性管理を行う．また，「意紙」は複数の表現を表すためのデータモデルなので，一つの「意紙」が同属性の永続オブジェクトを複数持つことはできない．

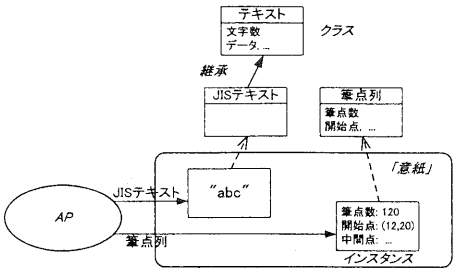

図1：「意紙」とクラス

2.5.2 永続オブジェクトのライフサイクル

コンストラクタを呼ぶことで，永続オブジェクトが生成され，デストラクタを呼ぶことで，永続オブジェクトが削除される．マップされた永続オブジェクトやそのポインタは複数のタスクから共有することができる（図2）．

このようにコンストラクタ，デストラクタを利用することで，永続オブジェクトを言語の型システムから利用できるように写像する作業を隠蔽している．

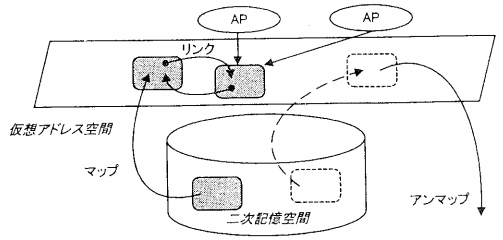

図2：オブジェクトのマップ

2.5.3 トランザクション

複数タスク間の操作からの整合性を確保するために「意紙」では，データベースで一般的に用いられているトランザクションを提供している．コミットを実行することで，永続オブジェクトに対する変更が二次記憶へ書き出される．また，アボートを実行することで，永続オブジェクトに対する変更が無効化される．アボートを用意することで，異常終了処理や条件判定に対して変更を無効化することができる（図3）．

ワンレベルストアでは，ポインタによってデータ間を相互に関連付けているため，一部のデータの破壊がシステム全体に波及する可能性があり，信頼性に問題がある．そこで，履歴などによって信頼性を確保する研究がされている［2］．「意紙」では，このトランザクション単位で履歴を取る方法が考えられる．

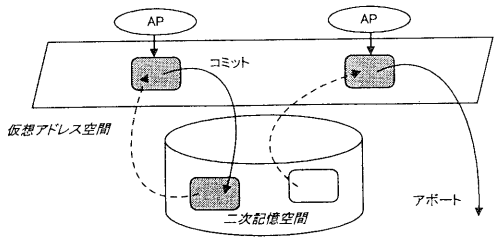

図3：オブジェクトのトランザクション

2.5.4 トラバース

永続オブジェクト間の関連性はポインタで記述することが可能であり，ファイルのようにデータのアクセスに対してハンドルを取得する必要はなく，ポインタをトラバースすることで，永続オブジェクトにアクセスすることができる．つまり，オブジェク

トの使用開始を明示的に記述する必要はない．

3.「意紙」管理機構の設計

3.1 属性管理

すべての属性と手続きはシステムに登録する必要があり，属性に対する手続きは関数表（図 4）によって管理している．プログラムからはこの関数表を介した間接参照により手続きを実行時にダイナミックリンクする．関数表は継承のような階層構造ではなく，フラットな属性の名前空間があり，属性ごとに手続き名の可変長リストを持つ構造になっている．手続きの実行は属性名と手続き名の組で指定する．このように言語内で静的にリンクを確定するのではなく，関数表を介すことで，再コンパイルすることなく動的に属性を追加，削除することができる．

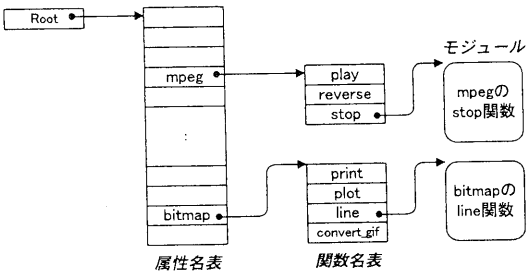

図 4： 関数表

「意紙」の多態性を実現する機構が属性変換機構である．「意紙」は複数の永続オブジェクトのインスタンスを包含しているが，属性変換機構は，それらの相互変換を提供する．例えば，図 1 を例にして，筆点列のインスタンスだけ存在する場合に，アプリケーションから JIS テキストのインスタンスを要求された場合，属性変換機構が JIS テキストのインスタンスを生成する．属性変換機構を実現するために，「意紙」内にどんなオブジェクトが存在するかというメタ情報を持つルートオブジェクトが存在する．ルートオブジェクトはプログラマからは隠蔽され，明示的に操作する必要はない．

3.2 オブジェクトの永続化

3.2.1 ポインタ形式

オブジェクト識別子としてポインタを利用する．ポインタ形式として，二次記憶空間と仮想アドレス空間上で同一のポインタを利用する方式と，二次記憶空間上では異なる形式のポインタ（永続ポインタ）とポインタを必要に応じて変換して利用する方式（Pointer Swizzling [3]）が考えられる．前者の場合は，変換のオーバヘッドが不要であるが，仮想アドレス空間より大きな二次記憶空間上のデータを扱うことができない．そこで，本機構では，後者を選択した．また，フラグメンテーションによるオブジェクトの再配置を容易にしたり，保護の単位とするために，セグメントを利用し，2 次元アドレス空間を提供する．

3.2.2 ポインタ変換機構

二次記憶空間に対する操作を明示的に記述することを不要にするために，ポインタをトラバースした時点でシステムが自動的に未参照のオブジェクトを仮想記憶空間にマップするポインタ変換機構を提供する．ポインタ変換は，セグメンテーションフォールト時とページング時に行う．このように MMU の機構を利用することで，ポインタ変換のオーバヘッドを削減することができる．

(1) セグメンテーションフォールト時

セグメンテーションフォールト時の動作を図 5 に示す．仮想アドレス空間上にはすでにオブジェクト A とオブジェクト C というオブジェクトがマップされている．オブジェクト A からオブジェクト B という未参照オブジェクトへのリンク参照が発生したとき，オブジェクト B のセグメント ID は永続セグメント ID であり，実際にそのようなセグメント ID は存在しないので，セグメンテーションフォールトが発生する．フォールトハンドラは，ストレージサーバに対してオブジェクト B のマップを要求し，マップ成功時にはオブジェクト A 中の永続ポインタをポインタに書き変え，処理を再開する．

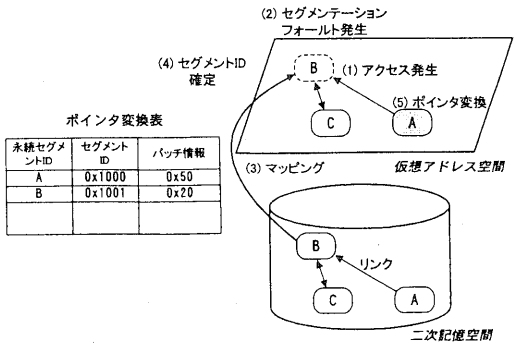

図 5： セグメンテーションフォールト時の処理

永続ポインタとポインタの対応はシステムがポインタ変換表として管理する．コミット時には，ポイ

ンタを永続ポインタに書き変えて二次記憶空間に格納することになる．そこで，ポインタ変換表にはどのオブジェクトのどのポインタを変換したのか，オブジェクト内のオフセットを記憶しておく必要がある．

(2) ページング時

仮想アドレス空間にマップされたオブジェクトは，実際にアクセスされ，ページング時が発生した時点に二次記憶空間から読み込まれる．

すべてのポインタ変換はセグメンテーションフォールト時に解決することができるが，フォールトのオーバヘッドは大きいので，すでにマップされたオブジェクトへの参照はセグメンテーションフォールトを起こす前の，ページング時に解決することで，フォールト数を減らすことができる．セグメンテーションフォールト後には，対象オブジェクトは仮想アドレス空間へマップされており，ポインタ変換表が出力されているので，ポインタがマップ済みであるかどうか判定することができる．そこで，ページフォールト時にページを二次記憶からロードすると同時に，ページ内のポインタを変換すればよい．

4.「意紙」管理機構の実装と評価

4.1 システムの全体構成

本機構を PC/AT 互換機（Pentium 166MHz，メモリ 80MB，IDE 6.4GB）で動作する OS/omicron 第 4 版（以下，V4）[4] 上に実装した．本機構の全体構成を図 6 に示す．

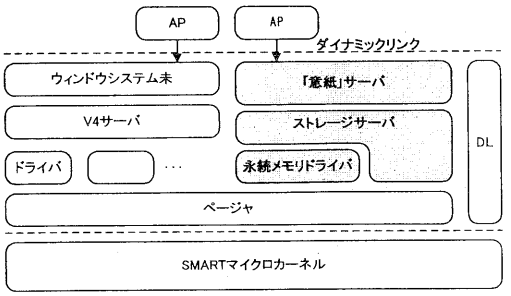

図 6： 全体構成

V4 はマイクロカーネル構成を採用しており，本機構は名前や属性を管理する「意紙」サーバ，永続オブジェクトを管理するストレージサーバとから構成される．また，V4 では，すべてのデバイスはメモリとして抽象化され，フォールト処理はデバイスドライバによって定義することができる [5]．そこで，セグメンテーションフォールト時のハンドラは，永続メモリドライバとして実装した．

4.2 「意紙」サーバ

関数表は，V4 におけるリンケージテーブルを利用することで実現した．リンケージテーブルはダイナミックリンカが利用する外部シンボル名の参照表であり，システムで統一された資源管理表として利用されている．

属性や手続きのリンケージテーブルへの登録は，コンパイル時に言語処理系が型情報を出力できれば，その型情報を利用することができるが，まだ型情報の出力は未実装である．したがって，プログラマが登録する必要がある．

リンケージテーブルに登録された手続きをロードするのにかかる時間を測定した．結果を表 1 に示す．一番最初に手続きを呼出した場合は，二次記憶から仮想アドレス空間へモジュールをロードする必要があるので約 22ms かかる．しかし，2 回目以降の呼出しはすでに仮想アドレス空間上にモジュールが存在するので，リンケージテーブルの検索だけが必要となる．したがって，モジュールをタスク間で共用することは，モジュールのロード時間の削減になる．

表 1： モジュールのロード時間

主記憶から	82.14 μs
二次記憶から	22,289.16 μs

4.3 ストレージサーバ

V4 では，単一 2 次元アドレス空間を採用しており，ポインタはセグメント ID 32 ビット，オフセット 32 ビットで表される．しかし，x86 で有効なセグメント ID は 13 ビットだけである．これでは，8k 個のセグメントしか生成できない．そこで，セグメント ID の 32 ビットすべてを利用した形式を永続ポインタとし，セグメンテーションフォールト時にポインタ変換することにした．二つのポインタは共に 64 ビット長であり，オブジェクトサイズに変更はなく，ポインタ変換時のオーバヘッドも小さくなる．

セグメンテーションフォールト時のポインタ変換にかかる時間を図 7 に示す．マップ時間の約 25% がポインタ変換のオーバヘッドである．したがって，マップ時にすべてのポインタ参照を解決する場合と比較して，ポインタの 25%以上が未参照なら高速であると言える．さらに，セグメンテーションフォールトのオーバヘッドがかかるのは最初の一回だけであり，参照が多い場合ほどセグメンテーショ

ンフォールトの影響は小さい．したがって，セグメンテーションフォールトによるオーバヘッドは問題にはならない．

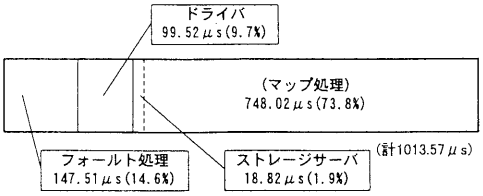

図7： セグメンテーションフォールト時のポインタ変換時間

5. 関連研究

ObjectStore [6] などのオブジェクト指向データベースは，オブジェクト指向言語をベース言語に利用し，永続オブジェクトを一時オブジェクトと透過にアクセスすることができる．オブジェクト指向データベースの多態性はベース言語に依存する．C++ がベース言語の場合は，仮想関数を利用することで多態性を実現できるが，リンクはコンパイル時に静的に決まるので，実行時にデータ型を動的に追加したりすることはできない．本機構では，システムが属性変換機構を提供することで言語に依存しない多態性を提供している．

また，Grasshopper [7] は，言語のデータ型と直交的に扱える永続記憶を提供する OS である．ObjectStore，Grasshopper 共に，ポインタ変換に Pointer Swizzling を利用している．多重アドレス空間で，Pointer Swizzling を利用する場合は，タスク間でポインタを含んだデータを共有することが困難である．本機構では，データ間の関連をポインタで表現するため，タスク間でポインタを共有することが必要である．V4 では，単一 2 次元アドレス空間をカーネル，言語処理系レベルから支援しているため，タスク間でのポインタの共有は容易に実現できる．

6. おわりに

本稿では，多態的表現を可能にする永続オブジェクト管理機構の設計と実現について述べた．本研究の成果を次に挙げる．

・一つの実体に対して複数の表現形式を持つオブジェクトを扱うために，多態的な永続オブジェクト管理機構を提供した
・ワンレベルストア環境を提供したことによって，永続記憶へのアクセスが容易になった

今後の課題は，言語処理系における型情報の出力を実現し，本機構を用いたアプリケーションを作成して，その有効性を評価することなどが挙げられる．

参考文献

[1] 高野了成，佐藤元信，早川栄一，並木美太郎，高橋延匡： “OS/omicron 第 4 版のデータ管理機構「意紙」サーバにおけるリンク機構の設計と実現”，情報処理学会研究会報告，98-OS-77，pp.191-196，1998.

[2] 横田大輔，高野了成，早川栄一，並木美太郎，高橋延匡： “ワンレベルストア上での開発を指向した履歴管理機構の設計と実装”，第 8 回コンピュータシステムシンポジウム論文集，Vol.97，No.8，pp.149-156，1997.

[3] J.Eliot B.Moss： “Working with Persistent Objects： To Swizzle or Not to Swizzle”，IEEE Transactions on Software Engineering，Vol.18，No.8，pp.657-673，1992.

[4] Eiichi Hayakawa，Tomoyuki Morinaga，Yasushi Kato，Kazuaki Nakajima，Mitarou Namiki，Nobumasa Takahashi： “OS/omicron V4： An Operating System for Handwriting Interfaces”，In Proceedings of the sixth International Conference on Human-Computer Interaction (HCI International’95)，pp.537-542，1995.

[5] 佐藤元信，森永智之，早川栄一，並木美太郎，高橋延匡： “OS/omicron 第 4 版のデバイス管理におけるシーケンシャルデバイス管理法式の拡張”，情報処理学会第 7 回コンピュータシステムシンポジウム論文集，Vol.96，No.7，pp.179-186，1996.

[6] C. Lamb，G. Landis，J. Orenstein，and D. Weinreb： “The ObjectStore database system”，Communication of the ACM，Vol.34，No.10，pp.50-63，Oct. 1991.

[7] Alan Dearle，at el. ： “Grasshopper： An orthogonally persistent operating system”，Technical report，Department of Computer Science，University of Adelaide，1994.