０９－開－１０

データベース構築促進及び技術開発に関する報告書

やわらかい分散オブジェクトシステムに関する調査研究

平成10年３月

財団法人　データベース振興センター

委託先　株式会社シネ・ジャーナルプロダクション

KEIRIN

この事業は、競輪の補助金を受けて実施したものです。

序

データベースは、わが国の情報化の進展上、重要な役割を果たすものと期待されている。今後、データベースの普及により、わが国において健全な高度情報化社会の形成が期待される。さらに海外に対して提供可能なデータベースの整備は、国際的な情報化への貢献および自由な情報流通の確保の観点からも必要である。しかしながら、現在わが国で流通しているデータベースの中でわが国独自のものは１／３にすぎないのが現状であり、わが国データベースサービスひいてはバランスある情報産業の健全な発展を図るためには、わが国独自のデータベースの構築およびデータベース関連技術の研究開発を強力に促進し、データベースの拡充を図る必要がある。

このような要請に応えるため、(財) データベース振興センターでは日本自転車振興会から機械工業振興資金の交付を受けて、データベースの構築および技術開発について民間企業、団体等に対して委託事業を実施している。委託事業の内容は、社会的、経済的、国際的に重要で、また地域および産業の発展の促進に寄与すると考えられているデータベースの構築とデータベース作成の効率化、流通の促進、利用の円滑化・容易化などに関係したソフトウェア技術・ハードウェア技術である。

本事業の推進に当って、当財団に学識経験者の方々で構成されるデータベース構築・技術開発促進委員会（委員長　東海大学教授　上條史彦氏）を設置している。

この「やわらかい分散オブジェクトシステムに関する調査研究」は平成９年度のデータベースの構築促進および技術開発促進事業として、当財団が株式会社シネ・ジャーナルプロダクションに対して委託実施した課題の一つである。この成果が、データベースに興味をお持ちの方々や諸分野の皆様方のお役に立てば幸いである。

なお、平成９年度データベースの構築促進および技術開発促進事業で実施した課題は次表のとおりである。

平成10年３月

財団法人　データベース振興センター

平成９年度　データベース構築・技術開発促進委託課題一覧

分　野		課　題　名	委　託　先
社　会	1	インターネット型先進材料DB活用プログラムの開発	(財) 次世代金属・複合材料研究開発協会
	2	インターネットを利用したイベント関連情報に関するデータベースの構築	(社) 日本イベント産業振興協会
	3	オーサリング型地図付地域ガイドデータのプロトタイプ構築	(財) 地図情報センター
	4	高齢者住宅介護情報のデータベース構築	(株) フォワード
	5	筑波研究学園都市研究便覧インターネット対応化事業	(株) 筑波出版会
	6	建築行政指導要綱のHTMLデータベース構築	日本建築法令（株）
	7	中小小売業のための商品仕入れ情報データベースプロトタイプ構築	(財) 店舗システム協会
地域活性化	8	新聞記事・画像データベース構築	琉球新報社
技　術	9	イメージファイリングの効率的活用を目指す書誌情報データベース検索技術の構築	(株) 会議録研究所
	10	やわらかい分散オブジェクトシステムに関する調査研究	(株) シネ・ジャーナルプロダクション

目　　次

1. はじめに

ギガビットLAN、高速インターネット（Internet II, III）等の開発が進み、マルチメディアデータを高速に通信できるようになってきている。一方、Windows, Unix等のオペレーティングシステムとTCP/IPの通信プロトコルが業界標準となり、CORBAに代表されるミドルウェアの標準化が主要な課題となってきている。CORBAでは、オブジェクト指向概念を基本として、データベースシステム等の資源をオブジェクトとしてモデル化し、オブジェクト間の標準的な通信手段を提供しようとするものである。こうした分散システムでは、オブジェクト間の相互通信に加えて、利用者の要求の変化、多様化に柔軟に適応できることと、オブジェクトの故障、ネットワークの構成の変化、負荷の変動等のシステム環境の変化にも適応できることが必要となる。こうしたシステムを、『やわらかい』分散システムとする。

このために、こうした利用者要求の変化とシステム環境の変化に対して、システムが自律的に自動的に適応できる機構についての調査研究を行う。ここでは、CORBAに準拠した分散オブジェクトシステムについて考える。本調査研究により、分散オブジェクトシステムのやわらかくするための方法について研究する。

(1) システムアーキテクチャ

まず、分散オブジェクトシステムのアーキテクチャを明確にする。ここでは、CORBA に準拠したシステムを考え、こうしたシステムの変化としてどのようなものがあるかを考える。また、システム環境の変化と利用者要求の変化に対して適応できる機構として、エージェントシステムを考える。

(2) オブジェクトの QoS

利用者から見て、オブジェクトは、データ構造とこれに対する操作演算を提供するブラックボックスである。オブジェクトがどのようなサービスを提供するかとともに、どのようなサービス品質（QoS：Quality of Service）を提供するかが重要である。ここでは、オブジェクトの提供する QoS を分類し、体系化する。オブジェクトの変化をオブジェクトの提供する QoS の変化としてモデル化する。特に、ビデオ、音声等のマルチメディアデータを提供するオブジェクトについて考察する。

(3) QoS に基づいたやわらかいオブジェクト

オブジェクトは、データ構造とこれに対する操作演算（メソッド）が一体化したものである。利用者要求の変化とシステム環境の変化に対して、自律的に適応する機構として、オブジェクトの提供する QoS を保証するための演算間の関係について考える。例えば、あるオブジェクトがアプリケーションの要求する QoS を提供できなくなったとき、全く同じオブジェクトでなくても要求する QoS を提供しているオブジェクトによりサービスを提供することが考えられる。こうした QoS に基づいたオブジェクト間の関係について考察する。

Window, Unix 等のオペレーティングシステムと TCP／IP の通信ネットワークが共通の基盤技術として業界標準となっている。こうしたインフラを用いて、データベースシステム、ワープロ等の多種多様なミドルウェアのインターオペラビリティを提供することが課題となり、このためにオブジェクト指向概念に基づいた CORBA が開発されてきている。本調査研究では、CORBAによりオブジェクト間のインターオペラビリティが提供された分散オブジェクトシステムを、利用者要求とシステム環境の変化に対して、自律的に適応させるための機構を検討する。特に、本調査研究では、エージェント技術を用いた『やわらかさ』を提供する機構についての研究を行うものである。分散したオブジェクト間のインターオペラビリティについての研究は CORBA 等で行われているが、システムを利用者要求の変化とシステム環境の変化に自律的に適応させる機構についての研究はなされていない。本調査研究により、より利用者が容易に利用できるための基盤技術を提供できる。

本報告書の第2章では、分散システムの基本となるオブジェクト指向システムについて述べる。第3章では、システムのモデルについて述べる。第4章では、オブジェクトのサービス品質（QoS）と、これに基づいたオブジェクトのやわらかさについて考える。第5章では、QoS 概念に基づいたオブジェクトの多重化方式について論じる。

2. オブジェクト指向概念

オブジェクト指向システムの基本的な概念について考える。オブジェクト指向システムは、以下の基本的な概念に基づいている。

【オブジェクト指向システムの基本概念】

①オブジェクト（object）
②メッセージパッシング（mesage passing）
③カプセル化（encupslation）と情報隠蔽
④クラス（class）とインスタンス（instance）
⑤階層構造－is_a と part_of
⑥継承（inheritance）
⑦ポリモルフィズム（plolymorphism）

2.1 オブジェクト

オブジェクト指向システムの基本となるオブジェクトの概念について、まず考える。オブジェクトとは、データ構造とこれを操作する演算とを一体化させたものである。オブジェクトの提供する演算をメソッド（method）という。利用者は、メソッドを利用してデータ構造を操作できる。この意味で、オブジェクトは、抽象データ型と同様な概念を有している。

【オブジェクト】オブジェクトは、以下が一体化した概念である。

①内部状態（データ構造）
②これに対する操作演算〔メソッド（method）〕

オブジェクト指向システムでは、システム内のすべての要素をオブジェクトとして考える。オブジェクトの値（データの値）を操作することに加えて、オブジェクト自体の生成、消滅といった操作が必要となる。車をオブジェクトとすると、車種、排気量等の値の変更と同時に、車が新しく生産されることによりオブジェクトが生成されるといったオブジェクト自体の操作が必要となる。このためには、オブジェクトが同じかどうかを考えることが重要となる。各オブジェクトは、システム内で、これを一意に識別する識別子を持っている。これをオブジェクト識別子（object identifier）という。オブジェクト指向システムでは、『同じ』という点について以下の二点を区別する必要がある。

①同一性（identical）
②同値性（equivalent）

二つのオブジェクトが同一であるとは、二つが同じものである、即ち、オブジェクト識別子が同じことである。これに対して、同値であるとは、オブジェクトの持つ値が同じことである。例えば、二つの車のオブジェクト C と D の排気量の値が同じで

あるとき、C と D は排気量について同値である。従来のプログラムでは、単に、変数の値が同じかどうかという、同値性のみをあつかっていた。これに対して、オブジェクト指向システムでは、同値性に加えて、オブジェクト間の同一性も考える必要がある。このことから、オブジェクト指向に対して、従来のプログラムを値指向ともいう。

【オブジェクトの同一性】 二つのオブジェクト O_1 と O_2 の識別子が同一ならば、O_1 と O_2 は同一である。O_1 と O_2 の値が同一であるとき、O_1 と O_2 は同値である。

【深同一性（deep equality）】 二つのオブジェクト O_1 と O_2 が以下の条件を満足するとき、O_1 と O_2 は深同一（同値）である。

①O_1 と O_2 は原子的オブジェクトであり、同一の値を持つ。

②O_1 と O_2 が集合オブジェクト $\{O_{11}, ..., O_{1n}\}$, $\{O_{21}, ..., O_{2m}\}$ であり、集合の要素 O_1 と O_{2i} が一対一対応し、対応する要素 O_{1i} と O_{2i} が深同一である。

③O_1 と O_2 が組オブジェクト $\{O_{11}, ..., O_{1n}\}$ $\{O_{21}, ..., O_{2m}\}$ であり、組の要素 O_{1i} と O_{2i} が一対一対応し、対応する要素 O_{1i} と O_{2i} が深同一である。

[プログラミングのスタイル]

①値指向：従来のシステム。オブジェクトとその値が区別されていない。

②オブジェクト指向：オブジェクト指向システム。実体としてのオブジェクトとその値を区別する。

2.2 メッセージパッシング

次に、オブジェクトの動作について考える。オブジェクト指向システムでは、オブジェクトにメソッドの実行を依頼することにより、計算が行われる。オブジェクトに、メソッドの実行を依頼することにより計算を行うことを、メッセージパッシング（message passing）という。実行の依頼を行うものを、メッセージという。メッセージは、オブジェクトに実行を依頼するメソッドを指定する。この指定を行うものを、メッセージのセレクタまたは選択子（selector）という。

【メッセージパッシング】

①オブジェクトは、メッセージを受信すると、メッセージのセレクタが指示するメソッドを実行する。

②オブジェクトは、内部状態を持ち、状態により受け付けられるメッセージを受信し、メッセージのセレクタが示すメソッドを実行し、内部状態を変化させる。

例えば、push、pop といったメソッドを提供するオブジェクトであるスタック（stack）に対して、選択子〈push〉を持ったメッセージを送信するとする。このとき、stack の最大値以下であれば、このメッセージを受け付けて、メソッド push を実行する〔図2.1〕。これはちょうど、分散システムにおけるクライアントサーバモデルのシステムにおいて、クライアントがサーバに仕事を依頼することと同じである。ここでは、サーバがオブジェクトであり、メソッドがサーバの演算である。

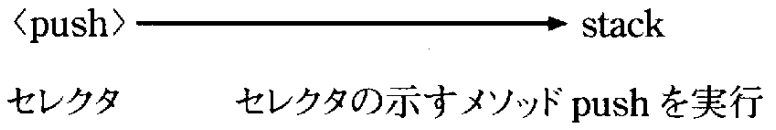


図2.1：メッセージ送信とセレクタ

メッセージパッシングは、分散システムにおけるクライアントサーバモデルのシステムにおいて、クライアントがサーバに仕事を依頼することと同じである。ここでは、サーバがオブジェクトであり、メソッドがサーバの演算である。計算方法を、リモートプロシージャ呼び出し（RPC: remote procedure call）という〔図2.2〕。

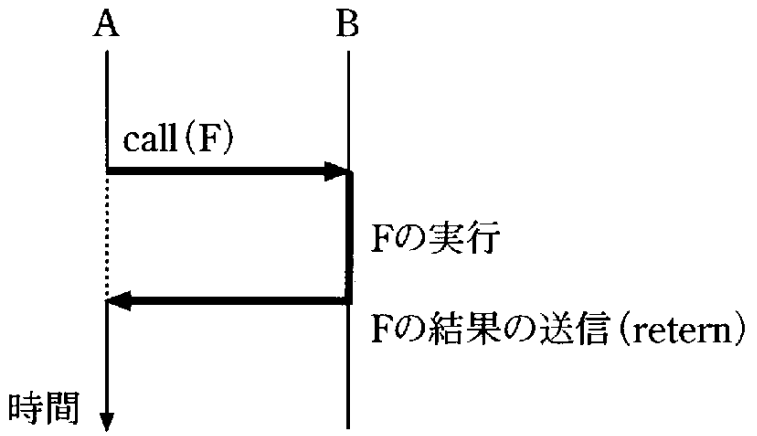


図2.2：RPC

従来のプログラムでは、他のモジュールは、関数呼び出しにより実行される。これは、関数が活性化、即ち、プログラムカウンタが、この関数呼び出し命令を示した時に、関数の引数となるデータを受け取って、関数が実行される。言い換えると、データが関数に送られることである。これに対して、オブジェクト指向システムでは、オブジェクトにメッセージを送信することにより、計算が行われる。

【実行方法】

①従来のシステム＝関数呼び出し〔図2.3〕。

②オブジェクト指向システム＝メッセージパッシング〔図2.4〕。

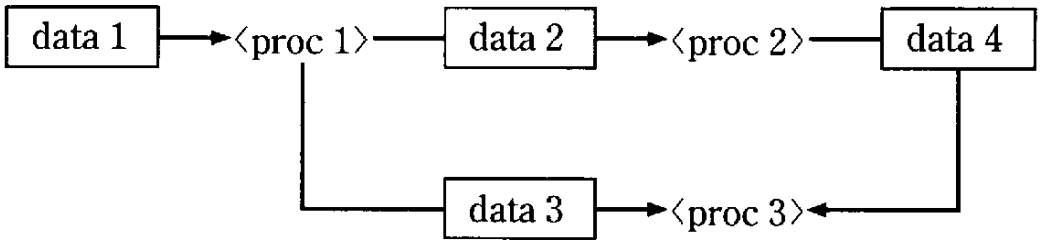


図2.3：関数呼び出し

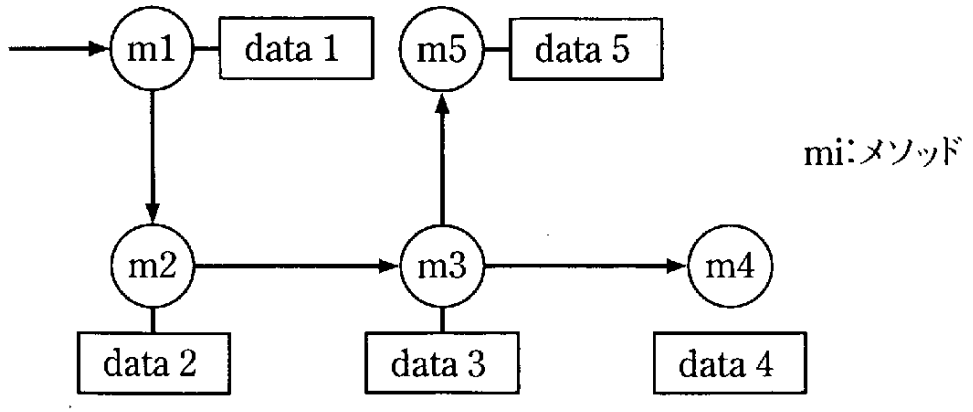


図2.4：メッセージパッシング

メッセージパッシングの実現方法について考える。従来の C. Fortran 等のプログラミング言語では、ソースプログラムは、コンパイルされ、リンクされて実行形式のプログラムが作られる。実行形式のプログラムでは、関数呼び出しは、ジャンプ命令に翻訳され、関数の番地にジャンプを行う。リンクされた後には、ジャンプ先の番地は、関数の番地であり、固定である。このために、実行時に、関数の呼び出し先を変更できない。これを、静的結合（static binding）という。これに対して、実行時に、関数名に対して、これの番地を決める方式を動的結合（dynamic binding）という。動的結合のプログラミングシステムでは、実行時に呼び出す関数を決めることができる。メッセージパッシングの実現方法として、この動的結合方式がよく用いられている。即ち、実行時にセレクタが示すメソッドをみつけ、これを実行する。

【実現方法の例】 例を〔図2.5〕に示す。オブジェクトは、主記憶内のセルであり、ここから、セレクタが示すメソッド名のテーブルへのポインタがある。このセレクタのテーブルを探索して、メソッド名をみつける。これからメソッドの実行コードをみつけ、実行する。

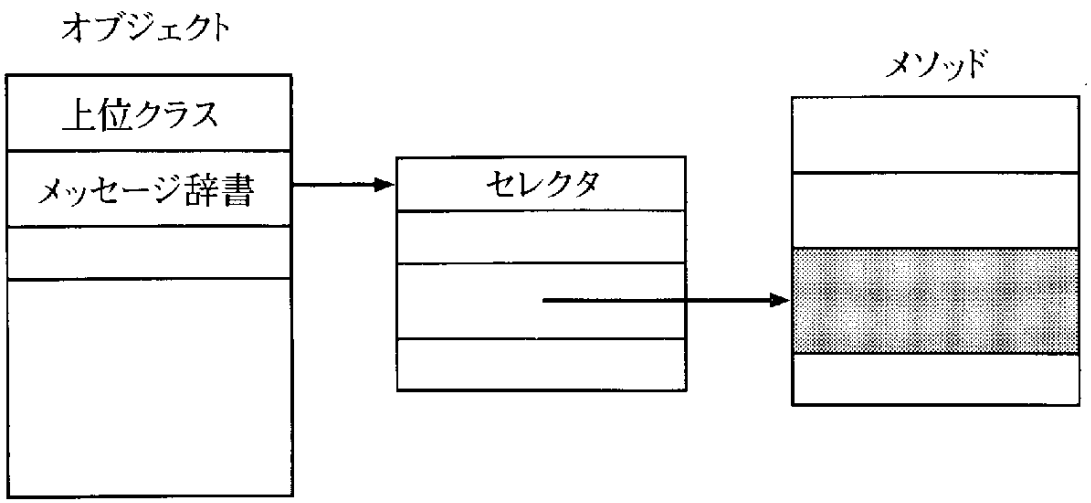


図2.5：メッセージパッシングの実現方法

2.3　カプセル化と情報隠蔽

データとこれに対する操作を行う手続きを一体化することを「カプセル化」という〔図2.6〕。これは、抽象データ型（ADT: abstract data type）と同じ概念である。オブジェクトのカプセル化を行うことにより、利用者はオブジェクトの操作演算を通してのみ、オブジェクト内のデータにアクセスすることができる。

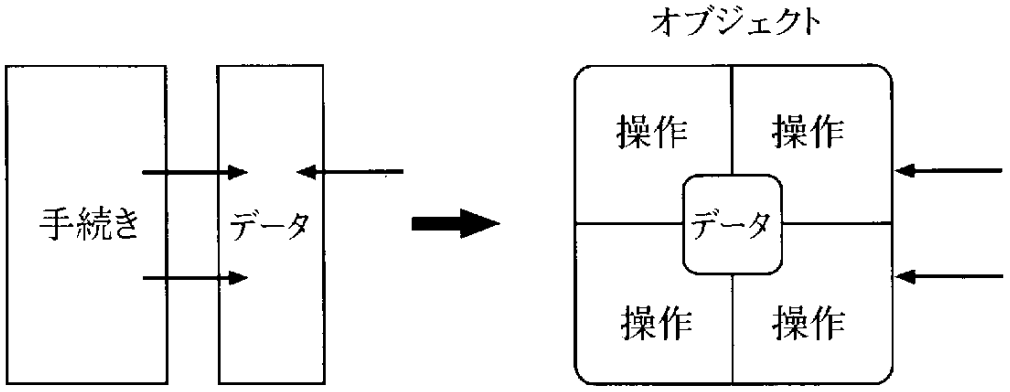


図2.6：カプセル化

情報隠蔽〔図2.7〕は、オブジェクトのカプセル化により実現される。オブジェクト内が情報隠蔽されることにより、利用者にとって以下の利点がある。

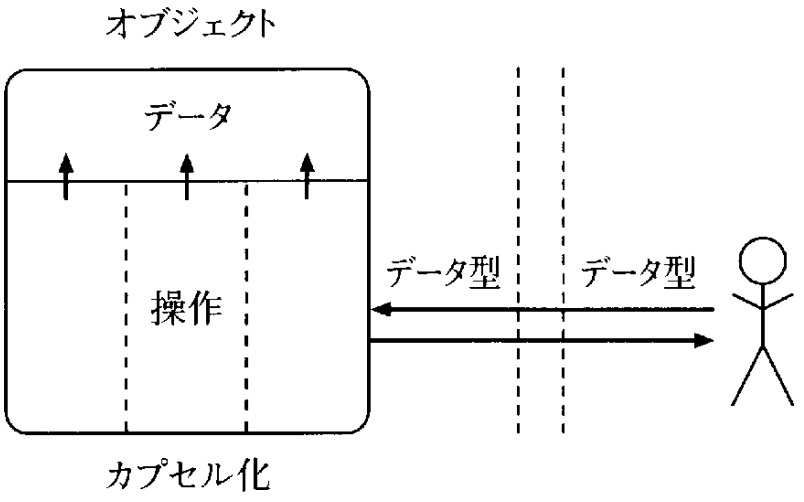


図2.7：情報隠蔽

①オブジェクトの実現方法と独立に、オブジェクトを利用できる。
②オブジェクトのデータ構造が変更となったり、メソッドの実現方法が変化しても、利用者には影響がない。

2.4 クラスとインスタンス

オブジェクトには、クラス（class）とインスタンス（instance）という二種がある。ここでは、クラスとインスタンスについて考える〔図2.8〕。クラスとは、共通の性質を持つオブジェクトの集合である。クラスは、こうしたオブジェクトの共通の性質を示すものである。データベースシステムのリレーショナルモデルで考えると、テーブルの属性の集合であるスキーマがクラスである。

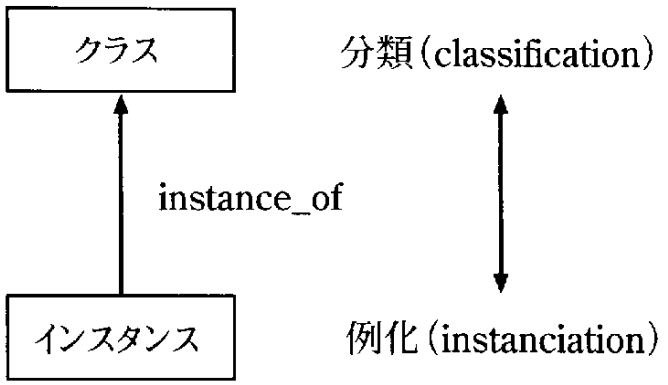


図2.8：instance_of

これに対して、クラスに属するオブジェクトを、このクラスのインスタンスという。リレーショナルモデルでは、属性集合であるスキーマに対して、属性に具体的に値を与えた行（またはレコード）の集合が、スキーマのインスタンスである。例えば、A、B、Cを人を示すオブジェクトとする。おのおの、名前（name）、年齢（age）についての値を持ち、A は 〈"a", 21〉、B は 〈"b", 20〉、C は 〈"b", 22〉 であるとする。このとき、クラス Man は、〈name, age〉 となる。〈"a", 21〉、〈"b", 20〉、〈"b", 22〉 は、クラス Man のインスタンスである。クラスを示すオブジェクトとそのクラスに属するインスタンスのオブジェクトの関係を instance_of という。

【例】〔図2.9〕は、クラスとしての学生と、インスタンスとしての個々の学生 A、B、C の間の instance_of の関係を示している。例えば、"A is an instance_of 学生" である。これを、図では、有向辺 "A→学生" で示す。

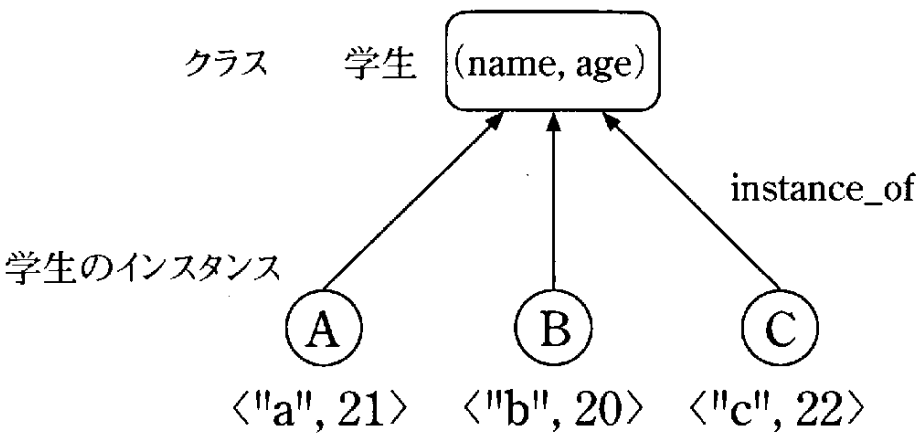


図2.9：クラスと例の関係

2.5 階層化

複雑なシステムを理解するためには、オブジェクトを階層化することが必要である。層化の方法としては、抽象化（abstraction）によるものと分割により行うものがある。前者は、is_a の関係といわれ、後者は、part_of の関係といわれる。オブジェクトを階層化することにより、システムの構築、保守、理解が容易となる。

2.5.1 汎化（generalization）

オブジェクト間の関係の一つとして、汎化（またはis_a）の関係がある。汎化とは、一つ以上のオブジェクトから、共通の性質を抽象化して、より一般的なオブジェクトを定義することである。これは、抽象化された上位の概念と、より特殊化された下位の概念の間の関係である。下位のクラスオブジェクトに対して、上位のオブジェクトを上位クラス（super class）という。又、上位クラスに対して、下位のクラスを副クラス（subclass）ともいう。下位のオブジェクトの集合から、上位のオブジェクトを導きだすことを、汎化という。一方、逆に、上位のオブジェクトから、下位のオブジェクトを導きだすことを、特殊化という〔図2.10〕。

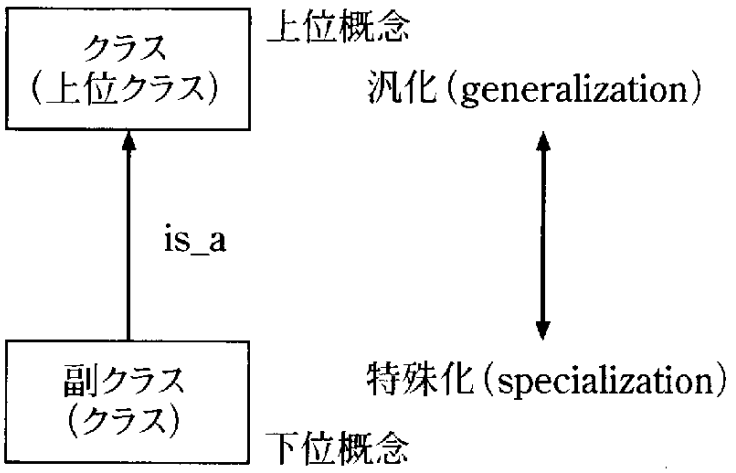


図2.10：クラス階層

〔図2.11〕に、コンピュータについてのクラス間の階層関係の例を示す。これは、ワークステーション（WS）とパーソナルコンピュータ（PC）に対して、コンピュータは、より一般的であり、WS と PC はコンピュータの一種であることを示している。すなわち、“Workstation is a computer”、“Personal computeris a computer” となる。

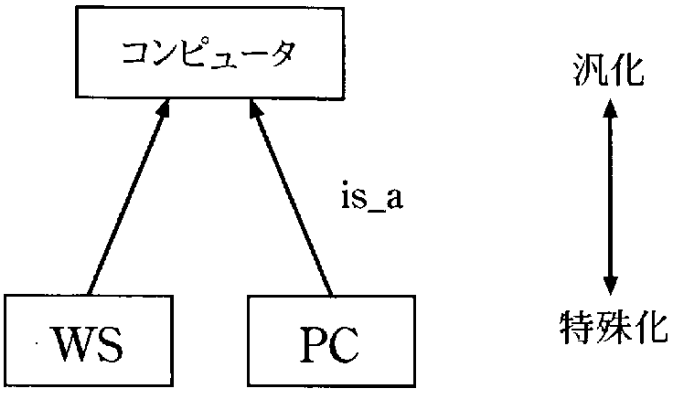


図2.11：階層化の例

2.5.2 集積化(aggregation)

階層化に加えて、一つの部品が他の複数の子部品から構成されているといった部品構成の関係を考える必要がある。オブジェクトが、他のオブジェクトを部品として構成されている場合である。これを集積化（aggregation）の関係という。集積化は、part_of ともいわれる関係である。部品を集めてオブジェクトを作ることを集積化、一つの部品から複数の子部品に分解してオブジェクトスを作ることを分割化という〔図2.12〕。

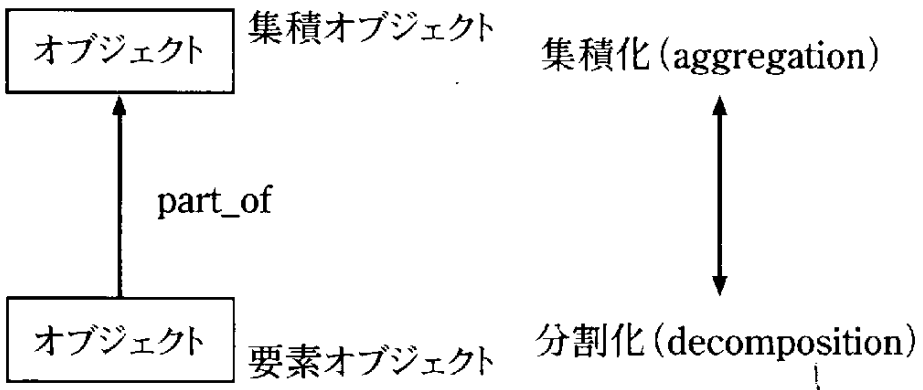


図2.12：集積化

【例】〔図2.13〕に集積化（part_of）の例を示す。図は、コンピュータというオブジェクトは、CPU、主記憶、二次記憶、I／O、バスといったオブジェクトから構成されることを示している。コンピュータと主記憶間には、汎化の関係はない。しかし、主記憶は、コンピュータの「一部」（part of）である。すなわち、“MEM is a part of コンピュータ”である。

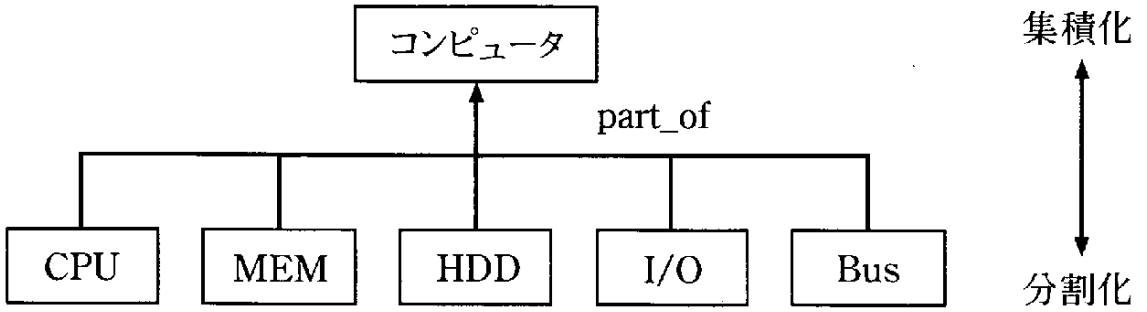


図2.13：集積関係

2.5.3 汎化と集積化

汎化（is_a）と集積化（part_of）の関係は、直行した概念である。〔図2.14〕に、コンピュータ、CPU、主記憶の間の汎化と集積化の関係を示す。縦方向に is_a の関係を示し、横方向に part_of の関係が示されている。オブジェクト WS_CPU は、CPU と is_a の関係にあると同時に、ワークステーションオブジェクト WS と part_of の関係にある。

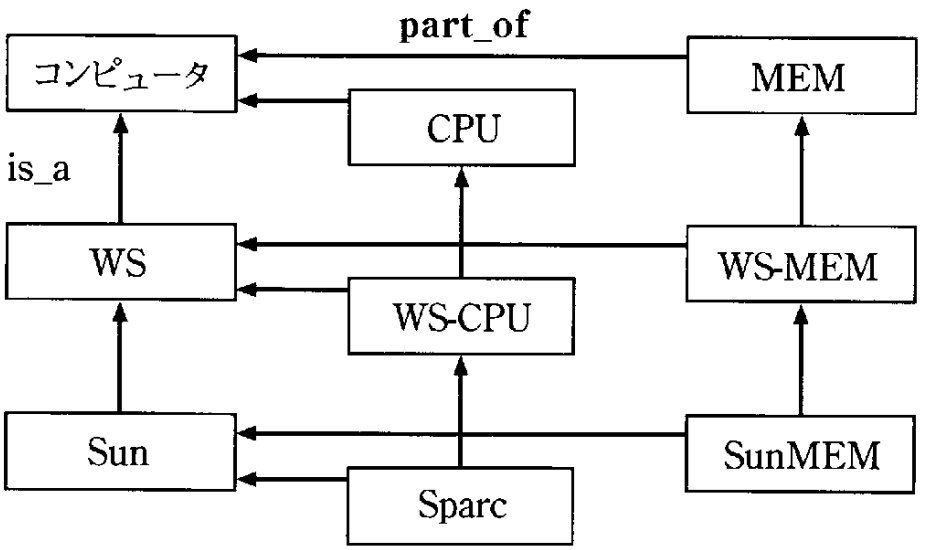


図2.14：is_a と part_of の関係

2.6 継承

ここまでに述べたように、オブジェクトが、汎化の階層構造を持つことから、上位と下位のクラスのオブジェクト間に、「継承（inheritance）」と呼ばれる関係がある。即ち、汎化の階層では、上位のオブジェクトの持つ性質は、下位のオブジェクトも持つ。例えば、ワークステーション（WS）がコンピュータ（computer）である（WS is_a computer）ということは、ワークステーションはコンピュータの性質を全て持つ

ことである。このように、上位のオブジェクトの持つ性質を、下位のオブジェクトが持つことを、上位オブジェクトの性質が下位のクラスに継承されるという。オブジェクトの性質としては、以下がある。

①データ構造。

②メソッド。

〔図2.15〕に、車の汎化の階層と、instance_of の関係の例を示す。ここで、上位から下位に継承される性質は、下位では記憶されない。例えば、Chevolet の工場(factory)についての情報は、このオブジェクトの中にはなく、上位のオブジェクト car 内にある。これに対して、上位にもあるが、下位にも同じ性質があるとき、下位のものが上位を「打ち消す(override)」という。Chevolet も car も、同じく type という属性を持つが、下位の Chevolet の type は、上位の car の type を打ち消している。

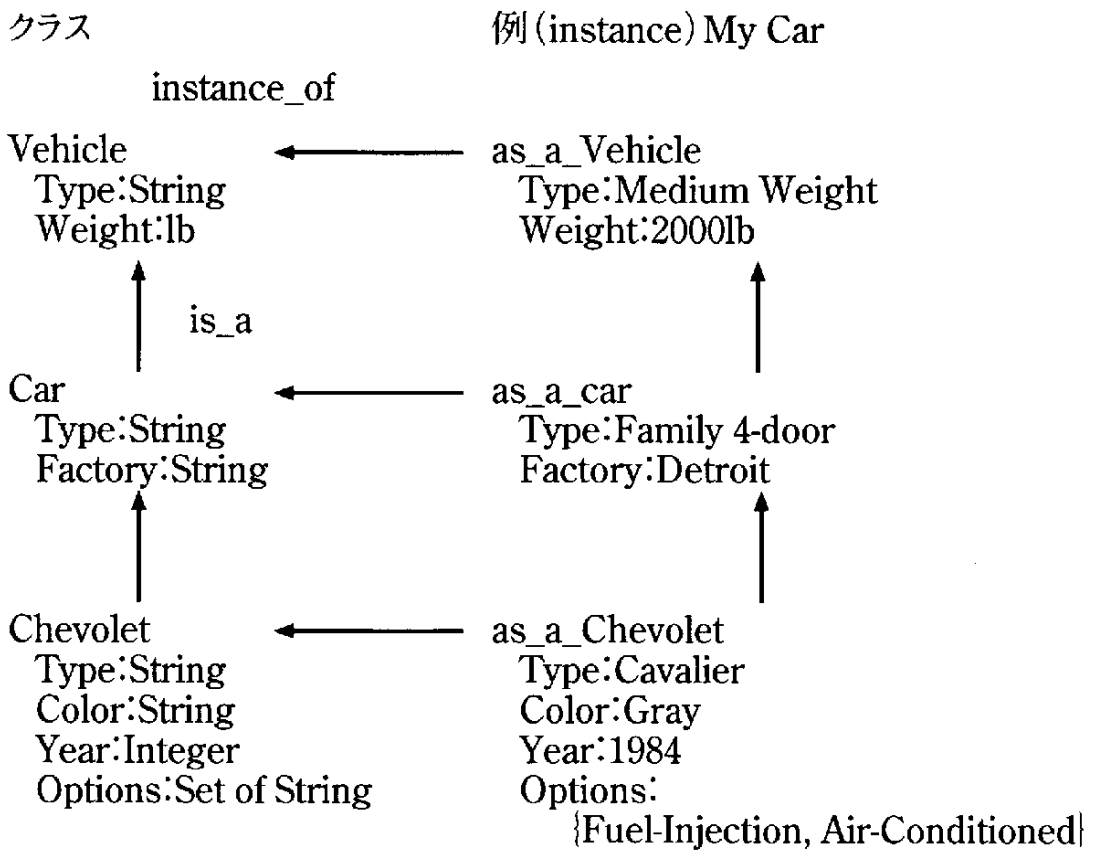


図2.15：継承階層の例

次に、〔図2.16〕に示す学生と会社員の例を考える。ここでは、学生を示すオブジェクト Student と会社員オブジェクト Employee (Emp) は、人間 Person であることを示している。即ち、Student is_a Person であり、Empis_a Person である。また、ある学生は、働いている (Student Worker) ことを示している。ここで、StudentWorker は、二つのクラス Emp と Student を上位のクラスとしているので、StudentWorker は、Emp と Student の両方の性質を継承している。このように、あ

るオブジェクトが複数の上位オブジェクトをもつときに、多重継承（multiple inheritance）されるという。このとき、StudentWorker の bonus は、Emp のものか、Student のものかという問題が生じる。Emp の bonus は給料に関したものであり、Student の bonus は成績に関している。このように、複数の上位クラスから、矛盾した性質を継承することを、競合（conflict）という。

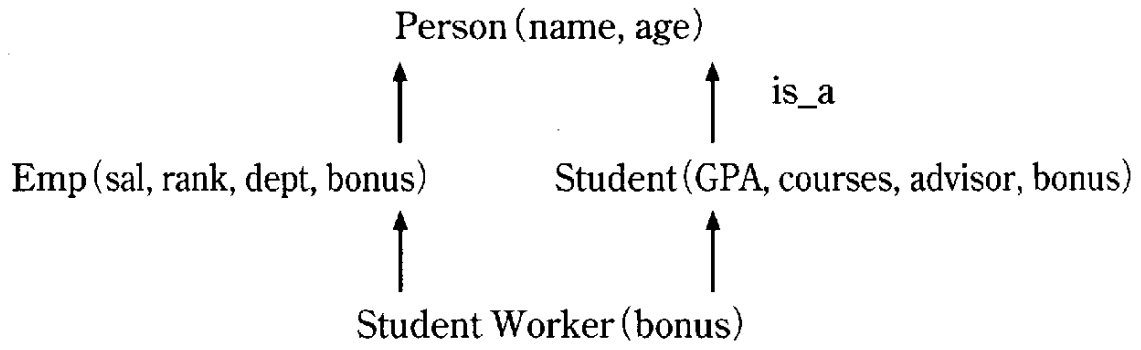


図2.16：多重継承の例

【多重継承】 あるクラスが複数の上位のクラス C1, ..., Cn を持ち、各上位クラス Ci から性質 pi を継承する場合である〔図2.17〕。

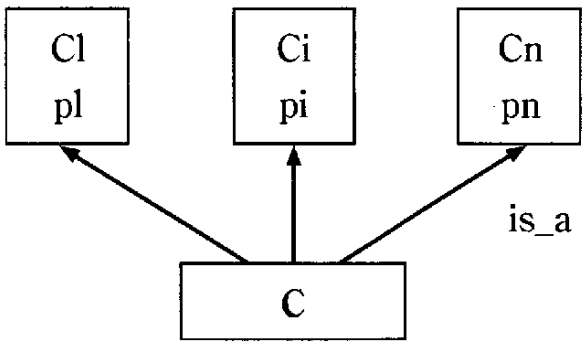


図2.17：多重継承

多重継承では、継承される性質が競合する（conflict）場合が問題となる。上位のクラス C1, ..., Cn に優先度を与えて、優先度の高いクラスの性質が継承されるものとする方法がよく行われている。また、クラス Cj から継承する性質を pj とする。このとき、p1, ..., pn が矛盾する場合が問題となる。この矛盾を例外を示すために利用することも行われている（Loops, Flavor）。一般に、クラス間の汎化の階層は、木構造よりは、束（lattice）の構造を持つ。

2.7 ポリモルフィズム

ポリモルフィズム（polymorphism）とは、異なったオブジェクトに対して、同じ名前のメソッドにより同様の操作を行えることである。例えば、オブジェクト File、Stack、Integer を考える。これらに対して、印刷を行うメソッドを print とする。同

名のメソッド print により、File、Stack、Integer のオブジェクトの内容をプリントできる。オブジェクト毎に異なった名前のメソッドを用いないで、同じ名前のメソッドにより操作できるために、オブジェクトの利用が容易となる。

【例】 メソッド print を用いて、種々のオブジェクトの印刷を行える。

print……x（integer）
print……y（real）
print……z（image）
print……w（file）

2.8 オブジェクト指向システムの特徴

これまでにみてきたオブジェクト指向システムの特徴をまとめると、以下のようになる。

【利点】 オブジェクト指向システムの利点として、以下がある。

①情報隠蔽：オブジェクトをどのように実現しているかとは独立に、オブジェクトを利用できる。オブジェクトの実現方法が変化しても、利用者には影響を与えない。

②データ抽象化：オブジェクトは、データ構造と手続き（メソッド）の一体化させたものである。利用者は、オブジェクトが提供するメソッドを通してのみオブジェクトを操作できる。オブジェクトを高信頼なものとすることにより、システム全体の信頼性を向上できる。

③継承：上位オブジェクトのデータ、メソッドを継承させることにより、ソフトウェア資源の再利用性を向上できる。ソフトウェアのモジュールの再利用である。

④動的結合：新しいオブジェクトを、容易に追加できる。これまでのプログラム言語を用いると、新しい関数を追加すると、再コンパイルし、リンクする必要がある。

【欠点】 オブジェクト指向システムの欠点としては、以下がある。

①実行速度：メッセージパッシングにより計算がおこなわれるために、実行速度が遅くなる。

②実現の複雑性：不要となったオブジェクトの記憶領域を解放するためのガーベッジコレクションの機構、動的結合の複雑な機構等が必要となる。

③大規模システム化が困難：二次記憶内のオブジェクトを操作すると、性能が低下してしまう。

3. システムモデル

3.1 システム構成

システムは信頼性のある通信網により相互接続された複数のオブジェクト o_1, ..., o_n により構成される。それぞれのオブジェクト o_i は、データと o_i を操作するための抽象演算の集合 op_{i1}, ..., op_{ili} によりカプセル化されている。

演算は o_i の状態を変化させ、あるデータをレスポンスとして出力する。$op_{ij}(s_i)$ を o_i の状態 s_i において、op_{ij} を使用した際に得られる o_i の状態であるとし、$[op_{ij}\ (s_i)]$ を $op_{ij}(s_i)$ によって得られるレスポンスであるとする。また、$op_{ij} \cdot op_{ik}$ は、op_{ik} が op_{ij} の後に実行されることを意味する。ここで、演算間の競合関係は以下のように定義されている：

全ての演算の op_{ij} と op_{ik} において、もし、o_i の状態 s_i にて $op_{ij} \circ op_{ik}(s_i) \neq op_{ik} \circ op_{ij}\ (s_i)$ であり、かつ、$[op_{ij}(s_i)] \neq [op_{ik} \circ op_{ij}(s_i)]$、もしくは $[op_{ij} \circ op_{ik}\ (s_i)] \neq [op_{ik}(s_i)]$ であるならば、op_{ij} は op_{ik} と競合関係にある。例えば、映画を再生するような *movie* オブジェクトにおいて、*record* という演算は *delete* という演算と競合関係にある。op_{ij} と op_{ik} に競合関係がないのであれば、op_{ij} と op_{ik} が交換可能であるなら、同じ状態と同じレスポンスが op_{ij} と op_{ik} の計算順序に関係なく得られる。本調査では競合関係を意味的なものであると仮定する。

3.2 マルチメディアオブジェクトでのQoS

本調査研究では、オブジェクトとしてマルチメディアオブジェクトを考える。オブジェクト o_i の QoS は 2 つの見方ができる。状態 s_i から得られる QoS である state QoS と、o_i によってサポートされる演算を通して得られる QoS である *operation* QoS である。例えば、〔図3.1〕に示すような、*display* という演算をもつ *video* というビデオオブジェクトを考える。o_i の状態 s_i は 30fps のビデオデータをサポートし、これは状態の QoSlQ（s_i）である。しかし、*display* は、s_i からビデオデータの視野 [*display*]（s_i）である 20fps でしか再生できない。これは演算を通して得られる QoS Q [*display* (s_i)] である。

ここで、Q（s_i）を o_i の状態 s_i の状態から得られる QoS であるとし、$Q(op_{ij})$ を演算 op_{ij} がサポートする QoS であるとする。o_i のQoSは、o_i の演算を通して得られる。Q [(op_{ij})] を状態 s_i にて op_{ij} を使用することにより得られる QoS であるとする〔図3.2〕。Q [(op_{ij})] は Q（s_i）と Q（op_{ij}）の最小となる。$\langle s_i \rangle$ を $\langle [op_{i1}(s_i)], ..., [op_{ili}(s_i)] \rangle$ とする。つまり s_i の視野である。$Q\langle (s_i) \rangle$ を、$\langle Q([op_{i1}\ (s_i)]), ..., Q[op_{ili}(s_i)]) \rangle$ と定義する。すなわち、演算を通して得られる QoS である。Q（s_i）は、利用者が演算を通して得られる o_i の QoS を示す。もし、o_i に o_i 内の全ての演算 op_{ik}

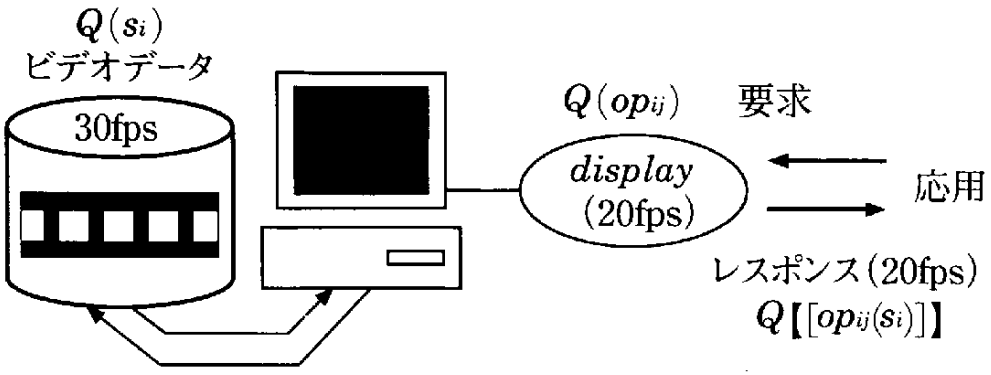


図3.1：ビデオオブジェクトのQoS

算 op_{ik} において、Q（[op_{ik}（$s_i \geq$ Q（[op_{ik}（s_j）]）なる演算 op_{ik} が存在するならば、そのときに限り、Q（〈s_i〉）は Q（〈s_j〉）を包含（Q（〈s_i〉）$\supseteq$ Q（〈s_j〉））する。o_i は、o_i によってサポートされる QoS の限界を o_i がサポートするQoS には、o_i の状態と o_i によってサポートされる演算の型に依存するため、範囲がある。max Q（o_i）を、o_i によってサポートされる最大 QoS とする。すなわち、Q（〈s_i〉）の最大である。また、min Q（o_i）を、o_i の最小 QoS であるとする。つまり、o_i の s_i において、min Q（o_i）≤ Q（〈s_i〉）≤ maxQ（o_i）である。

【定義】 全ての o_i の状態 s_i と o_i の状態 s_i において、（Q（〈s_i〉）$\supseteq$ Q（〈s_j〉））であるならば、そのときに限り、オブジェクト o_i は s_i を包含（$o_i \supseteq o_j$）する。

ここで、マルチメディアオブジェクト *movie* と *hypermovie* を考える。*movie* は低解像度のビデオデータ（120times×100pixels）と、そのビデオデータの再生を行なう *display* という演算をサポートする。*hypermovie* は高解像度のビデオデータ（160 times×120pixels）と、そのビデオデータを操作するさまざまな演算 *display* や *stop-motion, merge, divide* をサポートする。状態 s_{movi} は *movie* の低解像度のビデオデータを示す。$s_{hypermovie}$ は *hypermovie* の高解像度のビデオデータを示す。ここで、Q（$s_{hypermovie}$）≤ Q（s_{movie}）である。そして、*hypermovie* は *movie* よりも、高い品質のビデオデータと有益な演算をサポートしているため、*hypermovie* $\supseteq$ *movie* である。

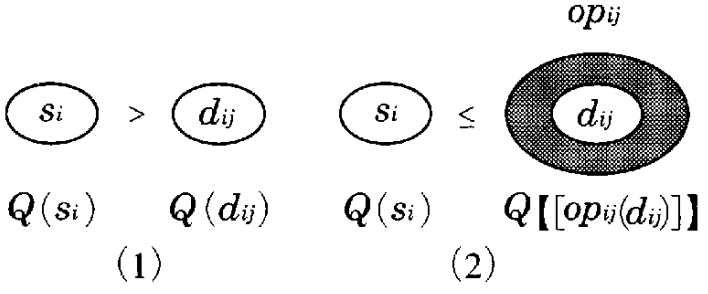


図3.2：演算を通して得られるQoS

4. やわらかいオブジェクト

オブジェクトは、データとこれに対する操作演算が一体化したものである。こうしたオブジェクトは外部の利用者もしくは他のオブジェクトに対して、あるサービスを提供する。ここでは、オブジェクトの提供するサービスの種類と品質（QoS：Quality of Service）について論じる。QoS に基づいたオブジェクトのやわらかさにつて論じる。

4.1 オブジェクトのサービス品質 (QoS)

オブジェクト o_i が提供するサービスは、o_i によって提供されている演算を実行することにより、得ることができる。こうしたオブジェクトの提供する様々なサービスの型は、パフォーマンス、コスト、安全性、可用性、信頼性などの属性によって特徴付けられる。例えば、信頼性は MTBF（*mean time between failures*）で表現される。可用性は MTBF／(MTBF＋MTTR) という式により与えられる。ここで MTTR は平均復旧時間 （*mean time to repair*）である。パフォーマンスは、スループットや反応にかかる時間などによって表現される。o_i によって支援されるサービスの品質（QoS）は、このような属性によって表現される。o_i と o_i の 2 つのオブジェクトが、同じ型のサービス提供するとしても、異なる QoS の値を提供するかもしれない。

QoS の構成は、a_i という属性の組で表され $\langle a_1, ..., a_m \rangle$ と表現される。属性 a_i のとりうる値の集合を定義域といい、$dom\,(a_i)$ と表現する。例えば、$dom\,(a_i)$ は、属性 a_i が解像度ならば、この属性の集合は画素（*pixel*）の集合となる。QoS の構成 $\langle a_i, ..., a_m \rangle$ のインスタンス $q\langle v_1, ..., v_m \rangle$ は、$dom\,(a_i)$ の直積によって表現される（$\langle v_1, ..., v_m \rangle \in dom\,(a_i) \times ... \times dom\,(a_m)$）。$dom\,(a_i)$ の値は $dom\,(a_i)^2$ に含まれるとき、支配関係があるといい、≤ を用いて表す。$dom\,(a_i)$ にある v_1, v_2の間に $v_1 \leq v_2$なる関係が成り立てば、v_1 は v_2 より良い QoS であることを示している。例えば、フレームレートという属性を考えれば、10 [fps] ≤ 20 [fps] というような支配関係が成り立つ。

ここで 2 つの QoS インスタンス q_1, q_2の関係について考える。ここで q_1 を $\langle v_{11}, ..., v_{1m} \rangle$ とし、q_2 を $\langle v_{21}, ..., v_{2m} \rangle$ とする。q_1 とq_2 に $v_{1i} \geq v_{2i}$ (i=1, ..., m) という関係が成り立つとき、q_1 は q_2 を全支配する（*totally dominate*）とし $q_1 \geq q_2$ と表す。

ここで、$q = \langle v_1, ..., v_m \rangle$ での属性 a_i の値v_i を $a_i\,(q)$ とする。さらに、$\langle a_1, ..., a_m \rangle$ の部分集合 $\langle b_1, ..., b_k \rangle$ をAとする。($b_i \in \{a_1, ..., a_m\}, k \leq m$)。$q$ の $[q]_A$は$w_i = b_i\,(g)$ $(i=1, ..., k)$ である組 $\langle w_1, ..., w_k \rangle$ である。$A_1 \cap A_2$ 内の属性 a に a $(q_1) \leq a\,(q_2)$ の関係があるなら、q_1 は q_2 を部分支配（*partially dominate*）するという〔図4.1〕。

また A_2 が A_1 の部分集合 ($A_1 \supseteq A_2$)であり、かつ q_1 が q_2 を部分支配するとき、q_1 は q_2 を包含するといい $q_1 \supseteq q_2$ と表す。また QoS の集合を Q と表す。Q 内の q_1 と q_2 の組ごとに、$q_1 \cap q_2$ もしくは $q_1 \cup q_2$ に最大下界と最小上界という i 関係がある

なら q_1 と q_2 には支配関係があり ≤ を用いて表現する。アプリケーションはオブジェクトに対していくつかの QoS を要求する。

本調査研究ではアプリケーションが要求する QoS を RoS（*requirement QoS*）とする。RoS は〈V_1, ..., V_k〉で示され、V_iは属性 a_iの値である。ここで、QoS（q = 〈v_1, ..., v_m〉）を支援するオブジェクト o が存在するものとする。また、要求を R で表し〈V_1, ..., V_k〉で表すことする。R の構成を A_R で表し、q の構成を A で表す。ここで、$A \supseteq A_R$で、かつ q が R を部分的支配するならば、q は R を包含する（$q \supseteq R$）。ここで、$q \supseteq R$のとき、アプリケーションはオブジェクト o から要求を満足するサービスを受けられる。

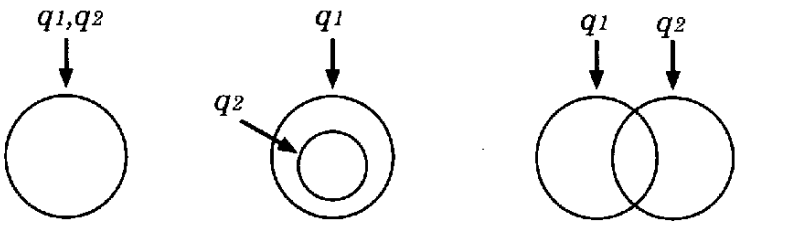


図4.1：QoS インスタンス間の関係

4.2 マルチメディアオブジェクト

本調査研究では、オブジェクトの対象としてマルチメディアオブジェクトについて考える。マルチメディアオブジェクトでは、QoS は 2 つの面を持つ。それは、"演算を通して得られるQoS" と "状態の QoS" である。例えば、高解像度のビデオデータをもつオブジェクト o_i と低解像度のビデオデータを持つオブジェクト o_j の 2 つを考える。このとき、o_i の状態 s_i が o_j の状態 s_i よりも良い場合を考える。o_i と o_j は圧縮されたデータをデコーダにより再生操作を支援しているとする。ここで、o_i と o_j が低いレベルのデコーダを支援しているとすれば、o_i は高解像度のビデオデータをもっていても o_i と o_j は同じ QoS を支援していることになる。このように、o_i のQoS は o_i の状態の QoS と o_i に対する演算を通して得られる QoS から構成されている。ここで、オブジェクト o_i の状態 s_i に注目する。s_i が支援する QoS を $Q(s_i)$ と表す。次に o_i の演算 op_{ij} が支援する QoS を $Q(op_{ij})$ と表す。o_i の QoS は o_i の演算を通して得ることができる。ここで、$Q([op_{ij}(s_i)])$ を s_i にて op_{ij} を通して得られる QoS を表すものとする。また $Q(\langle s_i \rangle)$ を、$\langle Q([op_{i1}(s_i)]), ..., Q([op_{ili}(s_i)]) \rangle$ と定義する。$Q(\langle s_i \rangle)$ は、オブジェクト o_i の QoS を示し、アプリケーションが演算を通してこれらを得ることができる QoS である［図4.2］。オブジェクト o_j の演算 op_{ik} ごとに、$Q([op_{ik}(s_i)]) \leq$ が成り立つなら、$Q(\langle s_i \rangle)$ は $Q(\langle s_j \rangle)$ を包含（$Q(\langle s_i \rangle) \supseteq Q(\langle s_i \rangle)$ する。ここで、$maxQ(o_i)$ をオブジェクト o_i の QoS の最大値を表すものとする。ま

た $minQ(o_i)$ をオブジェクト o_i の QoS の最小値を表すものとする。オブジェクト o_i には、$minQ(o_i) \leq Q(\langle s_i \rangle) \leq maxQ(o_i)$ なる関係が存在する。ここで、オブジェクト o_i, o_j 間の関係について以下のように定義する。

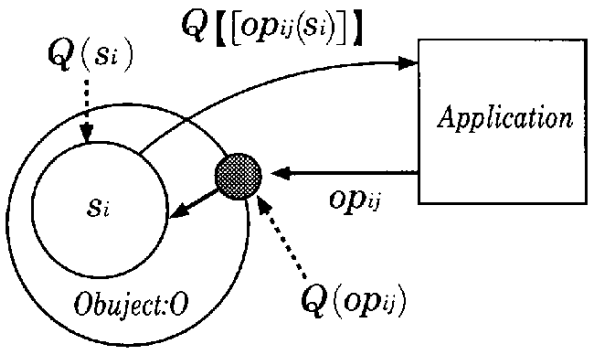


図4.2：オブジェクトの QoS

【定義】 オブジェクト o_j の状態 s_j、オブジェクト o_i の状態 s_i において、$Q(\langle s_i \rangle) \supseteq Q(\langle s_j \rangle)$ が成り立つなら、o_i は s_j を包含（$o_i \supseteq o_j$）する。

ここで、例として 2 つのマルチメディアオブジェクト *movie, hypermovie* を考える。*movie* オブジェクトは、*display* という演算を持ち、低解像度（120×100pixels）のビデオデータを支援する。一方、*hypermovie* は、*display* という演算の他に *stop-motion* や、*merge* という演算を持ち、高解像度（160×120pixels）のビデオデータをもつ。S_{movie} は、*movie* オブジェクトの状態を示し、ある映画 m の低解像度のビデオデータをもつ。また、$S_{hypermovie}$ は映画 m を含んだ高解像度のビデオデータをもつ状態を示す。この場合において、$Q(S_{hypermovie}) \leq Q(S_{movie})$ である。また、*285 hypermovie* は *movie* に比べて、より多くの種類の操作を支援する。*hypermovie* の *display* という演算は、通常の低解像度のムービーをモニターで表示している間、マルチウィンドウによって高解像度な画像をうつすことも可能であるとする。これより、$Q([display(S_{hypermovie})]) \leq Q([display(S_{movie})])$ である。したがって、*hypermovie* $\supseteq$ *movie* となり、*movie* よりもより実益のある、高い品質の画像を *hypermovie* は支援できる。

本調査研究では、マルチメディアデータの編集を行えるマルチメディアオブジェクト（以下 *ME*）について考える。*ME* システムにおいて、サービスは以下のような QoS 変数によって特徴づけられる。QoS 変数として考えられるものを以下に示す。

(1) CPUのスピード：MIPS
(2) 解像度：画素（pixel)数　（例）160×128画素
(3) 1秒間に送られるフレーム数　（例）10fps
(4) 色数：画素に用いられる色の数　（例）256色
(5) 音のサンプリング周期　（例）44.1kHz

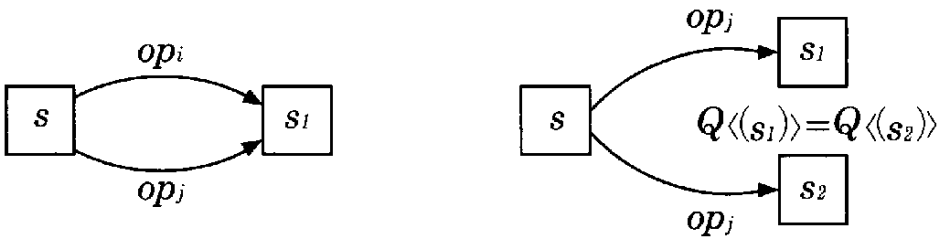
(6) 信頼性：MTBF
(7) 可用性：MTTR

4.3 同値性

本調査研究では、オブジェクト o によってサポートされる演算 op_i と op_j の間の同値関係について述べる。もし、o の状態 s において、$op_i(s)=op_j(s)$ かつ $[op_i(s)]=[op_j(s)]$ であるならば、そのときに限り、op_i は op_j と同値である〔図4.3 (1)〕。すなわち、op_i と op_j は、同じ出力結果だけでなく、o の状態も同じに変更する。例えば、*movie* というオブジェクトに *old-display* と *new-display* という 2 つのバージョンの *display* という演算があった場合について考える *new-display* は *old-display* と同様にビデオデータを再生できるが、*old-display* よりも高速に再生できるものとする。ここで、*new-display* は *old-display* と同値である。なぜなら、2 つの演算は同じビデオデータを再生し、かつ *movie* の状態を変化させないためである。しかし、それぞれの演算は異なるレベルの QoS をサポートしている。すなわち、*new-display* は *old-display* よりも、再生速度に関して優れている。

【定義】 オブジェクト o の状態 s において $Q(\langle op_i(s)\rangle)=Q(\langle op_j(s)\rangle)$ であるならば、そのときに限り、op_i と op_j は QoS 的に同値である。

すなわち、$op \circ op_i$ と $op \circ op_j$ (s) は、全ての演算 op に対して、同じ視野をサポートする〔図4.3 (2)〕。もし、$Q(\langle op_i(s)\rangle)=Q(\langle op_j(s)\rangle)$ であるならば、op_i と op_j は QoS 的に同値である。



(1) 同値性　　(2) QoS的同値性

図4.3：同値性のある演算

ここで R を、アプリケーションがオブジェクトに対して要求する RoS であるとする。もし、アプリケーションが表示速度に関して考慮しないのであれば、*new-display* を使用しても *old-display* を使用しても、アプリケーションから見ても問題はない。もし、2 つの演算が R を包含する QoS をサポートするのであれば、例え、Q (*old-display* $\langle S_{movie}\rangle$) $\neq Q$ (*new-display* $\langle S_{movie}\rangle$) であるとしても、2 つの演算は同値であると考えることができる。

【定義】 $Q(\langle op_i(s)\rangle) \cup Q(\langle op_j(s)\rangle) \supseteq R$ であるならば、そのときに限り、op_i と op_j は RoS 的に同値である〔図4.4〕。

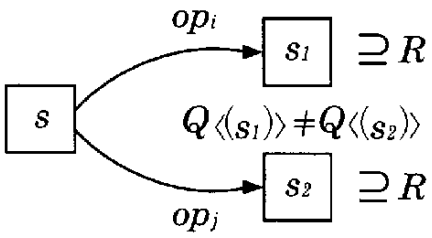
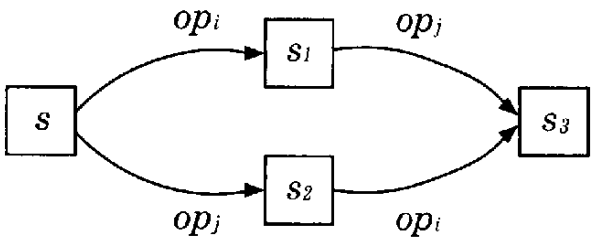


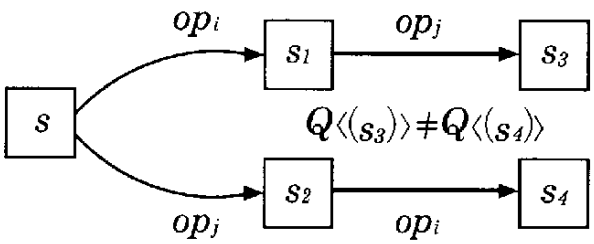
図4.4：RoS 的同値性

4.4 互換性

次に本調査研究では、オブジェクト o によってサポートされる 2 つの演算 op_i と op_j において、o の状態の一貫性を保つ実行順序について述べる。一般的にはもし、op_i の後に op_j を実行して得られる状態と、op_j の後に op_i を実行して得られる状態が異なるならば、op_i と op_j は競合関係にあるとしている。op_i と op_j が競合関係にないのであれば、op_i と op_j は互換である〔図4.5 (1)〕。



(1) 互換性



(2) QoS的互換性

図4.5：互換性のある演算

例えば、映画再生オブジェクト m が、広告と本来の内容で構成されているものとする。広告部分は m から *delete* という演算により削除される。ここで、アプリケーションは本来の内容にしか注目しないのであれば、アプリケーションにとって最初のバージョンと更新されたバージョンの違いは関係なくなる。すなわち、m の更新されたバージョンは、m の最初のバージョンと同じ QoS をサポートする。

本調査研究では、演算 op_i と op_j の間に QoS 的に互換である関係を定義する。

【定義】 オブジェクト o の状態 s において、$Q(\langle op_i \circ op_j(s)\rangle) = Q(\langle op_j \circ op_i(s)\rangle)$ であるならば、そのときに限り、op_i と op_j は QoS 的に互換である〔図4.5 (2)〕。

QoS 的互換性は対称性である。op_i と op_j が QoS 的に互換でないならば、op_i と op_j は QoS 的競合関係である。例えば、演算 *delete* は *movie* から、いくつかのフレームを削除するものとする。ビデオデータは低レベルデコーダである *display* を使用して再生されるものとする。ここで、アプリケーションは *delete* がビデオデータに対して使用されたにもかかわらず 低レベルなデコーダのために以前と同じ品質のビデオデータのように再生する場合について考える。つまり、*delete* と *display* は QoS的に互換である。

【定義】

$Q(\langle op_i(s)\rangle) \cup Q(\langle op_j \circ op_i(s)\rangle) \supseteq R$, $Q(\langle op_i \circ op_j(s)\rangle) \cup Q(\langle op_j(s)\rangle) \supseteq R$かつ $Q(\langle op_i \circ op_j(s)\rangle) \cup Q(\langle op_j \circ op_i(s)\rangle) \supseteq R$ であるならば、そのときに限り、op_i と op_j は RoS 的に互換である〔図4.6〕。

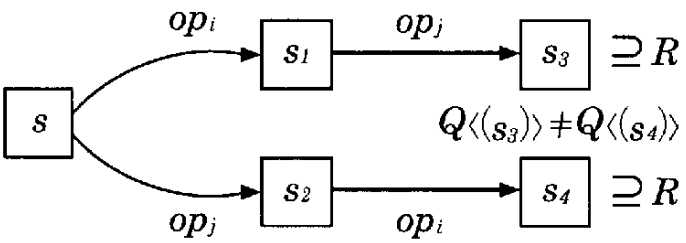


図4.6：QoS 的同値性

ここで、アプリケーションにとってビデオデータがカラーかモノクロであるかは問題でない場合について考える。*update* という演算がビデオデータをカラーからモノクロのバージョンへ変更してしまったとする。オブジェクト *movie* はカラーのビデオデータ m をサポートするものとする カラーのビデオデータは *display* によって再生される。すなわち、[*display* (m)] である。もし、*update* が m に対して実行されたとしても、m はモノクロとして再生されることになる。なぜなら、アプリケーションにとって m がカラーかモノクロかは関係なく、両方のバージョンともアプリケーションによって要求される QoS (RoS) を満足するからである。よって、Q([*display* (m)]) $\cup$ Q([*update* $\circ$ *display* (m)]) $\supseteq$ かつ Q(*display* $\circ$ *update* (m)) $=$ Q (*update* $\circ$ *display* (m)) である。しかし、2 つの演算はQ([*update* $\circ$ *display* (m)]) $\neq$ Q([*display* (m)]) であるため、QoS 的に互換でない。

4.5 補償

4.5.1 補償演算

マルチメディアアプリケーションにおいて、映画の再生デザインなどのように、アプリケーションは前に行った作業を取り消したい場合があるかもしれない。一般的な方法では、オブジェクト o_i はチェックポイントを取得する。そして、格納されたログ l_i によって、o_i は以前の一貫性のある状態へロールバックする。例えば、もし、o_i が障害を起したなら、o_i は、o_i 内の l_i に記録されたチェックポイント c_i へロールバックする。そして、再スタートする。多くのプロトコルにて、複数のオブジェクト間の一貫性のあるチェックポイントの取得方法や再スタート方法が議論されている。

他の方法として、前に実行された演算の影響を取り除くために、いくつかの演算を計算する方法がある。もし、オブジェクト o の状態 s において、$op_i \circ op_j(s) = s$ であるならば、op_j は op_i の補償演算である。$o\tilde{p}_i$ を op_i の補償演算であるとする。s' を、o の状態 s において op_i を実行したときに得られる状態であるとする。すなわち、$s' = op_i(s)$ である。ここでもし、$o\tilde{p}_i$ が s' において実行されたならば、oはロールバックすることができる。

例えば、*append* という演算は *delete* の補償演算である。ここで、連続する演算op_1, ..., op_m がオブジェクト o にて実行されたとする。この演算を取り消すためには、補償演算 $o\tilde{p}_m$, ..., $o\tilde{p}_1$ が実行される必要がある。すなわち、全ての演算 op において、$op \circ op_1 \circ ... \circ o\tilde{p}_m \circ ... \circ o\tilde{p}_1 = op$ である。

4.5.2 QoSに基づいた補償

o の状態 s と s' の対は、もし、s と s' が異なっていても、アプリケーションから見れば同値に見える場合があるものと考える。例えば、2 つの A と B という変数がある場合について考える。始め、A=100、B=50である状態 s_1 であったとする。その後、A と B が操作され、A=110、B=40という状態 s_2 になってしまったとする。もし、アプリケーションが A と B の総計にしか感心がないのであれば、s_2 は s_2 と同値であると考えることができる。

ここで、2 つのそれぞれ 2 時間の映画 A と B を持つマルチメディアオブジェクト ME がある場合について考える [図4.7]。この状態を s_1 とする。次に、A と B がマージされ、映画 C が生成されたとする。この状態を s_2 とする。そして、C が A' と B' の 2 つの映画に分割された場合について考える。A' と B' は共に 1 時間30分の映画であるとする。この状態を s_3 とする。映画 A と B はそれぞれ、広告と本来の内容を含んでいるものする。一方、A' と B' は、それぞれの本来の内容のみを含んでいるものとする。A と B の広告部分は映画 AB にマージされた場合について考える。ここで、状態 s_3 と s_1 は $Q(\langle s_3 \rangle) = Q(\langle s_1 \rangle)$ であるため QoS 的に同値である。つまり、*divide* は *merge* の補償演算である。

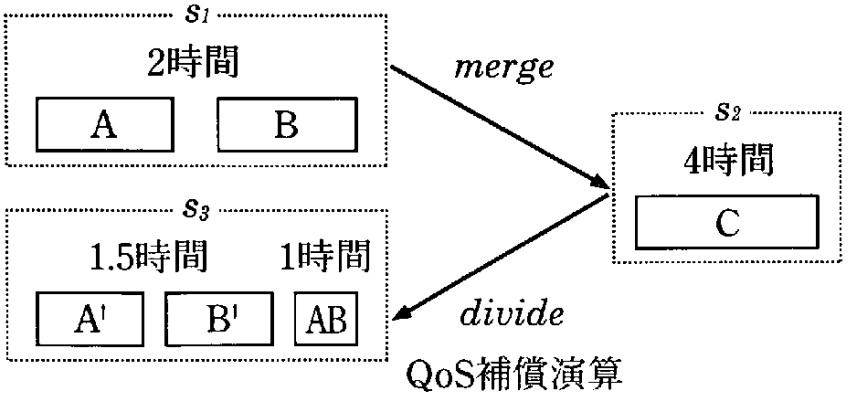
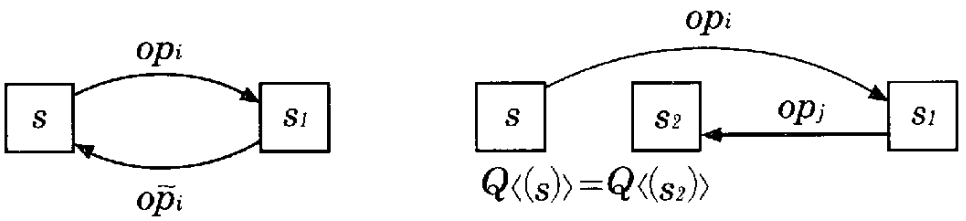


図4.7：QoS 補償演算

ここで、状態 s_1 が、オブジェクト o の状態 s にて演算 op_1 を実行したときに得られる状態であるとする。本調査研究では、s_1 から s へ o がロールバックする方法について考える。1 つの方法は、$op_i \cdot op_i(\tilde{s}) = s$ であることから、状態 s_1 にて、op_1 の補償演算 op_i を計算することである〔図4.8(1)〕。ここで、$op_i \cdot op_j(s) = s_2$ かつ $s \neq s_2$ であり、$Q(\langle s_2 \rangle) = Q(\langle s \rangle)$ となる演算 op_j が存在するものとする。s_2 は s と異なる。しかしながら、s_2 は s と QoS 的に同値である〔図4.8(2)〕。



(1) 補償演算　　(2) QoS補償演算

図4.8：補償演算

【定義】 オブジェクト o の状態 s において、$Q([op_i \circ op_j(s)]) = Q(\langle s \rangle)$ であるならば、そのときに限り、op_j は op_i の QoS 補償演算である。
op_i を op_i の QoS 補償演算であるとする。

〔図4.9〕において、映画 C が 2 つの映画 A と B をマージすることによって得られるものとする。s_1 を A と B が存在する状態を示すとし、C が存在する状態を s_1 であるとする。ここで、マルチメディアオブジェクト ME が、C を 3 つの映画 A'', B'', AB に分割する *divide2* という演算をサポートしている場合を考える。A'' と B'' は A と B の映画本来の内容であり、共にモノクロである。AB は、A と B の広告部分である。状態 s_3 を A'', B'', AB が存在する状態であるとする。s_1 と s_3 は等しくない。更に、A と B はカラーであるが、A'' と B'' はモノクロである。すなわち、$A \supseteq A'$ かつ $B \supseteq B'$ である。ここで、アプリケーションは RoS R として、たった今モノクロの再生を要求しているものとする。つまり、$Q(\langle s_3 \rangle) \supseteq R$ である。

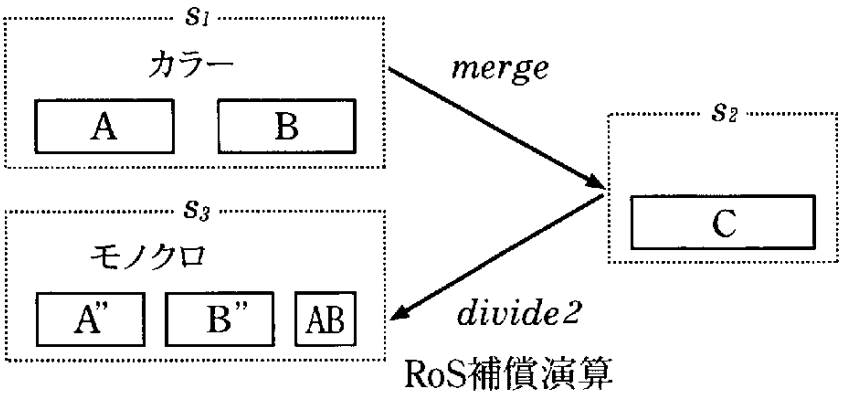


図4.9：RoS 補償演算

【定義】 オブジェクト o の状態 s において、$Q([op_i \cdot op_j(s)]) \cup Q(\langle s \rangle) \supseteq R$ であるならば、そのときに限り、op_i は op_j の RoS 補償演算である。
divide2 は *merge* の RoS 補償演算である。

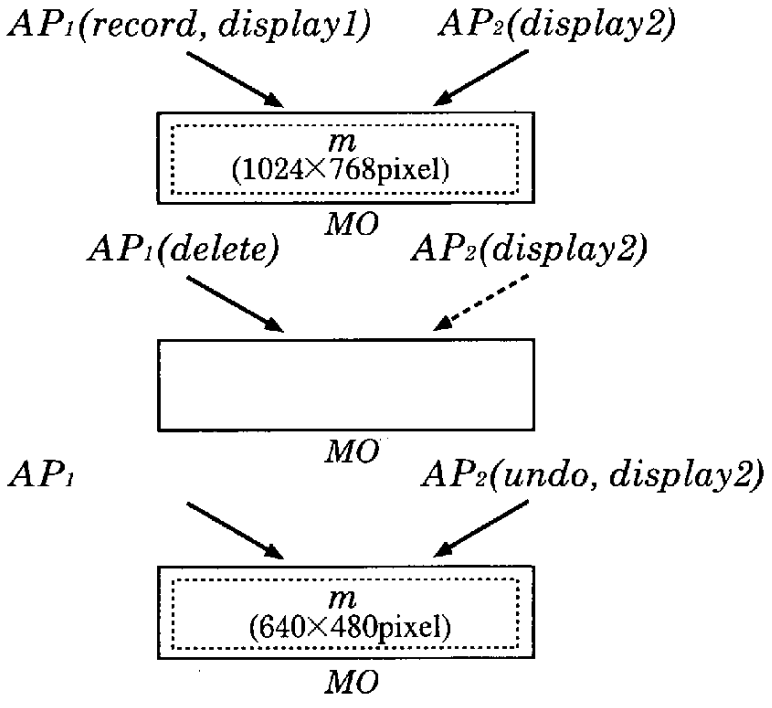


図4.10：医療オブジェクト

ここで、医療手術をサポートする *medical object* (*MO*) を考える〔図4.10〕。アプリケーション AP_1 が *MO* に、手術の内容をビデオデータ m として記録したとする。m は高解像度のビデオデータ（1024×768pixels）である。そして、AP_1 が演算 *display1* という演算を使用し、1024×768pixels にて m を再生させるとする。一方、アプリケーション AP_2 は、m を *display2* という演算を通し、640×480pixels にて再生させている。ここで、AP_1 が、m を削除した後に再び m を格納したい場合を考える。ビデオデータ m' は、*undo* により生成された解像度が640×480pixels である。

しかし、質は m より劣るが、AP_2 は m' を再生できる。つまり、*undo* は、*record* の RoS 補償演算である。m とまったく同じビデオデータを格納するよりも、*undo* によって m' を得るほうが短時間ですむ。

4.6 QoS を考慮した多重化

システムは複数のオブジェクト $o_1, ..., o_n$により構成される。一般的なシステムでは、オブジェクトは信頼性、可用性、性能向上のために多重化される。もし、それぞれのレプリカが同じデータと、それを操作する同じ演算をサポートしていても、異なるレベルの QoS をサポートするかもしれない。アプリケーションは、それが要求する QoS (RoS) を十分にサポートできるレプリカを使用してサービスを提供される。ここで、もし、レプリカが RoS よりもより良い QoS をサポートするのであれば、アプリケーションはどのレプリカを使用してもよい。

4.6.1 参照演算

始めに、アプリケーションがオブジェクト o_iから、状態 s_iにて、演算 op_{ij}を使用してスナップショット $[op_{ij}(s_i)]$ を取得したい場合を考える。o_i は、op_{ij} にて明記される制限を満足する $[op_{ij}(s_i)]$ だけでなく、十分な QoS である $Q([op_{ij}(s_i)])$ もサポートしなければならない。ここで、R をアプリケーションがオブジェクトから得るスナップショットの要求する QoS(RoS) であるとする。もし、$R \subseteq Q([op_{ij}(s_i)])$ を満足するような op_{ij}をサポートするオブジェクト o_iが存在するなら、アプリケーションは op_{ij}を使用しスナップショットデータを得るために o_iへアクセスすることが可能である。さもなければ、アプリケーションはどのオブジェクトへもアクセスできない。例えば、3 つのオブジェクト o_1, o_2, o_3があり、それぞれ、$Q([op_1(s_1)]) \subset R$, $Q([op_2(s_2)]) \supseteq R$, $Q([op_3(s_3)])$ Rである場合と考えるこの場合、o_2もしくは o_3がアプリケーションによって操作される。なぜなら、o_2と o_3は R を満足するからである [図 4.11]。もし、$Q([op_2(s_2)]) \supseteq Q([op_3(s_3)])$ であるならば o_2がアクセスされる対象となる。

ここで、マルチメディアオブジェクト ME を例として考える。ME_1 は、80×60pixels のビデオデータ m_1 と、それを80×60pixels にて再生できる演算 *display1* をサポートしているものとする。一方で、ME_2 は、120×100pixels のビデオデータ m_2 と、それを120×100pixels にて再生できる演算 *display2* をサポートしているとする。また、ME_3 は、160×120pixelsのビデオデータ m_3と、それを160×120pixels にて再生できる演算 *display3* をサポートしているものとする。そして、アプリケーションは、100×80pixels 以上のビデオデータの再生を望んでいる場合を考える。この場合、$Q([display1(m_1)])$=〈80×60pixel〉, $Q([display2(m_2)])$ =〈120×100pixel〉, $Q([display3(m_3)])$=〈160×120pixel〉, R 〈100×80pixel〉 である。つまり、ME_2と ME_3 がアプリケーションの要求するサービスをサポートすることができる。そして、

$Q([display2(m_2)]) \subset Q([display3(m_3)])$ であることから、アプリケーションは ME_3 からサービスを得ることができる。

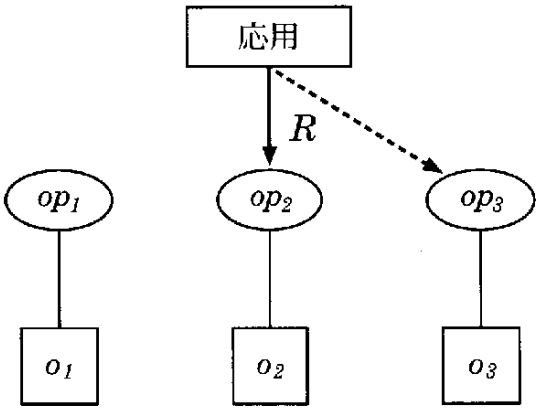


図4.11：QoS を考慮したレプリカ

ここで、それぞれ op_1 と op_2 をサポートするオブジェクト op_1 と op_2 が存在する場合を考える。アプリケーションは op_1 を通して o_1 へアクセスしている。もし、o_1 が障害を起した、もしくは o_1 が RoS R をサポートできなくなってしまったならば、アプリケーションはもはや o_1 を使用することはできない。ここでもし、o_2 がアプリケーションの満足する十分な QoS をサポートするのであれば、すなわち、o_2 の状態 o_2 において $Q([op_2(s_2)]) \supseteq R$ ならばアプリケーションは、o_1 に代わって o_2 へアクセスすることができる。

【定義】 オブジェクト o_j の状態 s_j と演算 op_j、およびオブジェクト o_i の状態 s_i と演算 op_i において、$Q([op_j(s_j)]) \supseteq Q([op_i(s_i)])$ であるならば、そのときに限り、o_j は o_i の QoS レプリカである。ここでもし、$Q([op_j(s_j)]) = Q([op_i(s_i)])$ であるならば、o_j は o_i の完全 QoS レプリカである。たとえ、2 つのオブジェクト o_1 と o_2 が同じデータと演算をサポートしていても、o_1, o_2 が異なる QoS をサポートしているのであれば、完全 QoS レプリカではない。$QR(o_i)$ は、o_i の QoS レプリカの集合を示すものとする。

【定義】 オブジェクト o_j の状態 s_j と演算 op_j、およびオブジェクト o_i の状態 s_i と演算 op_i において、$Q([op_j(s_j)]) \supseteq Q([op_i(s_i)])$ であるならば、そのときに限り、R において o_j は o_i の QoS レプリカである。

ここでもし、$Q([op_j(s_j)]) = Q([op_i(s_i)]) \supseteq R$ であるならば、o_j は o_i の完全 QoS レプリカである。$RR(o_i)$ は、o_i の RoS レプリカの集合を示すものとする。

4.6.2 更新演算

次に、*merge* や *divide* のように、オブジェクト o_i の状態を変化させる演算を考える。演算は o_i にデータを書き込むと考えることができる。ここで、W をシステム内へデータを格納する際にアプリケーションが要求する QoS (RoS) であるとする。o_i の RoS レプリカ $o_i, ..., o_n$ は変更されなければならない。レプリカ $o_i, ..., o_n$ 以下の規則

に従い操作される：

(1) $W \supset maxQ(o_i)$ であるならば、アプリケーションは o_i へ $maxQ(o_i)$ にて書き込みを行う。

(2) $minQ(o_i) \subseteq W \subseteq maxQ(o_i)$ であるならば、アプリケーションは o_i へ W にて書き込みを行う。

(3) $W \subset minQ(o_i)$ なるオブジェクト o_i が在するならば、アプリケーションはどのオブジェクト o_j $(j=1, ..., n)$ に対しても書き込みできない。

ここで、アプリケーションが 3 つのマルチメディアオブジェクト ME_1, ME_2, ME_3 へビデオデータを格納する場合を考える。アプリケーションは要求する QoS R として 25fps をそれぞれのオブジェクトへ要求するものとする。それぞれのオブジェクト ME_1 は表 1 に示す QoS $(Q\langle ME_i\rangle)$ をサポートする。この場合、アプリケーションは R を満足する 25fps にてビデオデータを格納することが可能である。なぜなら、それぞれのオブジェクトは $min(ME_i) \subseteq R$ を満足するからである。しかし、ME_3は、$max(ME_3)=\langle 15fps\rangle$ であるため、15fps にてビデオデータを格納することになる。ここでもし、R が 10fps であったならば、$min(ME_2)=\langle 15fps\rangle$ であるため、ME_2 へビデオデータを格納することができない [表 1]。

表1:最大 QoS と最小 QoS

オブジェクト ME_i	$max(ME_i)$	$min(ME_i)$
ME_1	25 fbs	10 fbs
ME_2	30 fbs	15 fbs
ME_3	15 fbs	5 fbs

ここで、マルチメディアオブジェクト ME を例にとって考える [図4.12] ME は複数のオブジェクト ME_1, ME_2, ME_3 により構成される。それぞれのオブジェクトは、それぞれ広告部分 Adv_1 と Adv_2 を 2 つの映画 M_1 と M_2 を保持している。そして、ME_1 と ME_3 は映画の広告部分と本来の内容を削除する演算 *delete1* をサポートしているものとする。一方で、ME_2 は、映画本来の内容のみを削除する演算 *delete2* をサポートしているものとする。ここで、アプリケーションが、それぞれのオブジェクトから M_1 を削除する場合を考える。ME_1 と ME_3 は Adv_2 を含む映画 M_2 のみを保持することになるが、ME_2 は、その他に Adv_1 も保持することになる。なぜなら、ME_2 は *delete2* によって操作されるからである。その結果、それぞれのオブジェクトは異なる状態を保持することになる。しかし、それぞれのオブジェクトによってサポートされる QoS は変化していない。すなわち、$Q(\langle S_{ME1}\rangle)=Q(\langle S_{ME2}\rangle)=Q(\langle S_{ME3}\rangle)$ である。つまり、*delete1* と *delete2* は QoS 的に同値である。それぞれのレプリカは、QoS 的に同値な演算によって操作されることを許す。

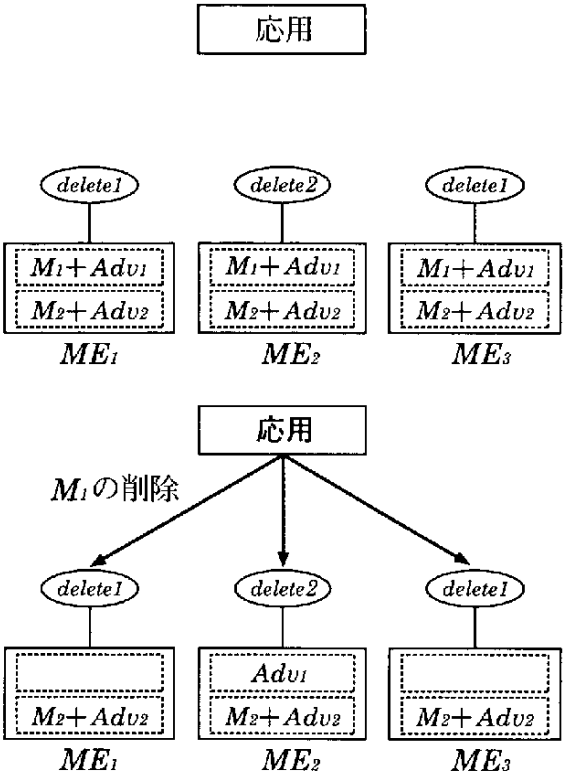


図4.12：Example

5. オブジェクトレプリカ間の一貫性

やわらかいオブジェクトを実現するためには、オブジェクトを多重化する必要がある。ここでは、QoSに基づいて多重化する方法について論じる。

5.1 同時実行制御

分散アプリケーションは、複数のオブジェクト（$o_1, ..., o_n$）の協調動作によって実現される。ここで、オブジェクト間のメッセージの交換は、通信網を通じて *RPC* (remote procedure call) によってのみ行われる。またオブジェクトは、限定された抽象演算によってのみ状態を操作される。システムは、トランザクションの並列実行、インターリーブ実行によるオブジェクトへの複数のアクセスに対し、オブジェクト間の相互の一貫性を保持しなければならない。

システム内のオブジェクトの一貫性を保持する方法として、同時実行制御というものがある。同時実行制御方式は、今まで多くの提案がされている。同時実行制御で有名な二相ロック方式（*2PL*）では、トランザクションは、オブジェクトに対し、演算を行う前にロックを行う。この後、連鎖的なアボートを防ぐために、トランザクションがコミットするときに、ロックをかけられた全てのオブジェクトに対してロックの解放を行う、厳格な二相ロック方式（strict 2PL）を採用しているシステムは多々ある。

厳格な二相ロック方式を用いない場合、ロックがトランザクションの終了前に開放されるために、システム内において連鎖的なアボートが起こる可能性がある。グループウェア（例 電子会議、電子授業）のような分散アプリケーションの問題点の１つとして、*2PL* を用いたときにオブジェクトが長時間ロックされることによりオブジェクトの使用効率が低下することが挙げられる。そこで現在、ロックによるオーバーヘッドを減少させるために、楽観的な（optimistic）同時実行制御が議論されている。従来の 2PL のような同時実行制御では、トランザクションが利用するオブジェクトは、他のトランザクションによって同時に利用され得ることを前提としている。このために、オブジェクトは、トランザクションが利用する前に必ずロックされる。しかし、複数のトランザクション間で同じオブジェクトが同時に利用されることが稀な場合には、ロックを行なうことは無駄となり、システムの性能の劣化につながる。楽観的な同時実行制御では、トランザクションはロックを行なわずにオブジェクトを利用する。オブジェクトの操作を行なった後に、他のトランザクションがこのオブジェクトを利用したかどうかを調べる。もし、どのトランザクションもオブジェクトを利用していないならば、コミットする。利用していれば、トランザクションはアボートする。

レプリカに対する楽観的な同時実行制御では、トランザクション T は、オブジェ

クトに対してロックを行わずにオブジェクトを操作する。トランザクションTが終了するとき、Tによって操作されているオブジェクト全てが、Tと競合する他のトランザクションに操作されないならば、Tはコミットする。

システム内のオブジェクトは、システムの信頼性、可用性、性能を向上させるために多重化される。また、移動型データベースシステムでは、非接続型演算を解決するために、固定型サーバから移動型クライアントへデータがキャッシングされる。オブジェクトが多重化されるとき、システム内の整合性を保つためにレプリカ間の一貫性を保つことが必要となる。

レプリカ間の一貫性を保つ方式として、Jingは、楽観的な二相ロック方式 (*O2PL*) を提案している。*O2PL*では、トランザクション Tが、*write* 演算を行う場合、Tにより、全てのレプリカが *read* モードでロックされ、ロックに成功したならばレプリカ上で演算を実行する。Tがコミットするとき、*read* モードのロックを *write* モードに変換し、全てのレプリカでロックモードの変換が成功したならば、Tはコミットする。変換が成功しなければ、Tはアボートする。

現在、分散アプリケーションは、オブジェクト指向の概念でモデル化されている。すなわち、アプリケーションはオブジェクトの集合としてモデル化され、オブジェクト間の協調動作により成り立っている。分散アプリケーションの標準的な枠組みとして*CORBA* が提案されている。これによりシステムはオブジェクトベースなシステムへと移行しつつあり、オブジェクトは今までのような低レベルの *read*、*write*に基づく演算ではなく、より抽象的な演算をサポートしている。例として、銀行オブジェクトにおいて、*deposit*（入金）、*withdraw*（出金）演算はオブジェクトにサポートされる抽象演算である。*deposit*、*withdraw* は、*read*、*write* 演算を組み合わせることにより実現される。オブジェクトは、そのオブジェクトがサポートする演算によってのみ操作が可能である。

本調査研究では、オブジェクトは演算 op_{it} によって操作する前に、op_{it} に対応する抽象的なモードでロックされる。抽象的なロックモードの例として、銀行オブジェクトにおいての *deposit* する。すなわち、オブジェクト o_i に対して、演算 op_1 が演算 op_2 と競合するならば、op_1 のロックモードと、op_2 のロックモードは競合する。我々は、レプリカ間の一貫性を保つ方式として、オブジェクトに基づくロック方式（*OBL*：*object based locking*）を提案する。

本調査研究では、ロックされるレプリカ数を、演算の競合確率とロックの強さに基づき決定する方式（OBL）をとる。すなわち、利用頻度が多く、競合する確率の少ない演算に対しては、初めにロックされるレプリカ数を少なくする方法をとる。

5.2 レプリカ

分散システムは、複数のオブジェクト（o_1, ..., o_n）が、通信ネットワークを通じて情報交換を行うことにより構成される。ここで、O をシステム内のオブジェクトの集

合 $O = \{o_1, ..., o_n\}$ $(n \geq 1)$ とする。通信ネットワーク N は、全てのオブジェクトの対 o_i、o_j 間では、信頼性のある双方向回線 (o_i, o_j) を用い、送信データにメッセージ紛失がないものとする。

トランザクション T は、オブジェクト o_i に演算 op_i を発行する。o_i は op_i を受け取り、o_i 上にて op_i を実行する。この時 op_i は、他のオブジェクト o_{ij} 上で、op_{ij} を呼び出すかもしれない。o_i は op_i の演算が実行終了後、実行結果を T に返す。T は、原子的な（atomic）演算の系列である。すなわち、トランザクションの実行するすべての演算が正常に完了したときのみ、トランザクションは正常終了（コミット）する。一つでも演算の実行に失敗した場合には、トランザクションは異常終了（アボート）する。この性質を、トランザクションの原子性という。T が呼び出す全ての演算が完全に終了、すなわちコミットするときのみ、T はコミットする。T により呼び出される演算 op_i は、op_i により呼び出される全ての演算が完了したときのみコミットする。T と同様に op_i も実行系列の原子的な単位である。このように、演算は入れ子型である場合がある。このような演算を入れ子型演算という。

各々のオブジェクト o_i は、o_i を操作する演算 $op_{i1}, ..., op_{iji}$ の集合 T_i をサポートしている。o_i と o_i をサポートする演算はカプセル化され、o_i はサポートする演算を通してのみ操作される。$op(s)$ は、状態 s の o_i に対して op を実行することにより、得られる状態を示している。ここで、$op_{iji} \circ op_{ijk}(s_i) = op_{ijk} \circ op_{iji}(s_i)$、すなわち、$o_i$ に対して op_{ij} を実行後、op_{ik} を実行した結果と、op_{ik} を実行後、op_{ij} を実行した結果が同様であるとき、op_{ij} と op_{ik} は伴立であるという。また op_{ij} と op_{ik} が非伴立であるならば、op_{ij} と op_{ik} は競合する（$op_{ij} \leftrightarrow op_{ik}$）という。$op_{ij}$ と op_{ik} が競合するならば、op_{ij} と op_{ik} を実行することにより得られる状態は実行順序により異なる。op_{ij} と競合する演算が、o_i 上で実行されている場合、演算が終了するまで、op_{ij} は実行するのを待つ。ここで、レプリカの状態を変化させる演算を unstable 演算と呼び、変化させない演算を安定（stable）演算と呼ぶ。

トランザクション T は、オブジェクト $o_1, ..., o_n$ を、それぞれに対応する演算 $op_{ij}, ..., op_{ili}$ によって操作する。op_i を受け取ると、o_i はロックモード $\mu(op_i)$ でロックされる。ここで、o_i のロックモードの集合を M_i とする。ロックモードの例として、ファイルシステムにおける *read*、*write* のロックモードは、$\mu(read) = rlock$、$\mu(write) = wlock$ で表される。銀行オブジェクトの抽象演算 *Deposit*、*Withdraw* のロックモードは、それぞれ *Dlock*、*Wlock* で表される。

ロックモード m_1 の演算 op_1 と m_2 の演算 op_2 が伴立であるとき、M_i 内で、m_1 と m_2 は伴立であるという。また、op_1 と op_2 が競合するとき、M_i 内で、m_1 と m_2 は競合するという。例えば、ファイルオブジェクトにおいては、*rlock* と *rlock* は伴立であり、銀行オブジェクトにおいては、*Dlock* と *Wlock* は伴立である。op_i と競合するロックモードにより o_i がロックされているならば、op_i はブロックされる。o_i が、$\mu(op_i)$ でロックされているときのみ、op_i は o_i 上で実行される。op_i の実行後、o_i にかけられたロックモード $\mu(op_i)$ は解放される。

システム内の信頼性、可用性、性能を向上するために、オブジェクト o_iは多重化される。多重化された o_i のレプリカの集合は、$R(o_i)=\{o_i^1, \ldots, o_i^{r_i}\}$ $(r_i \geq 1)$ と表される。

5.3 オブジェクトに基づくロック方式 (OBL：object-based locking)

o_i のレプリカ間の一貫性を保つ方式として、*OBL*（*object-based locking*）を提案する。

5.3.1 ロックモード

op_{ij} を実行する前に、オブジェクト o_i は、M_i 内の $\mu(op_{ij})$ でロックされる。ロックモード m でロックされている o_i 上で op_{ij} が実行されたとき、$\mu(op_{ij})$ が m と伴立であるならば、op_{ij} は、o_i で実行が開始される。$\mu(op_{ij})$ が m と競合するならば、o_i が m から開放されるまで、op_{ij} は実行を待たなければならない。ここで、o_i のロックモード mと競合するロックモードの集合を $C_i(m)=\{m' \mid m'$ は m と競合する$\}$ と表す。本調査研究では、各ロックモード間の伴立な関係は、対称的であるものとする。すなわち、m' と競合する関係にあるロックモードの集合内に m があるならば、m と競合する関係にあるロックモードの集合内に m' があることとする。

【定義】 M_i 内の任意の m_1、m_2 において、$C_i(m_1) \supseteq C_i(m_2)$ が成り立つ時、m_1 は m_2 よりも制限されるという。

【例1】 ファイルオブジェクト f の演算 *read*、*write* について考える。*rlock*、*wlock*を、それぞれ *read*、*write* のロックモードとする。*read* は、*write* と競合する関係であるので、*rlock* と *wlock* は競合する関係にある。*rlock* と *rlock* は競合せず、*rlock* と *wlock* は競合し、*wlock* と *wlock* も競合するものとする。*rlock* と、*wlock*が競合する関係を式で表すと、$C_f(rlock)=\{wlock\}$ と表せる。さらに、$C_f(wlock)=\{wlock, rlock\}$ と表せる。ここで、定義により、$C_f(rlock)$、$C_f(wlock)$ の部分集合であるので、*wlock* は、*rlock* よりも制限されるという。

【例2】 4 つの演算 *Deposit*（入金）、*Withdraw*（出金）、*Check*（残高参照）、*Audit*（検査）をサポートしている銀行オブジェクト B を考える。*Dlock*、*Wlock*、*Clock*、*Alock* をそれぞれ、*Deposit*、*Withdraw*、*Check*、*Audit*のロックモードであるとする。ここで、*Dlock* と *Wlock* は競合せず、*Clock* は *Dlock*、*Wlock* と競合するものとする。また、*Audit* は、B 内の口座と、会計を検査する演算であるとする。よって*Alock* は、*Dlock*、*Wlock* と競合し、*Clock* とは競合しないものとする。競合関係を式で示すと、$C_B(Dlock)=C_B(Wlock)=\{Alock, Clock\}$、$C_B(Clock)=C_B(Alock)=\{Dlock, Wlock\}$ と表せる。ここで式より、$C_B(Dlock) \cap C_B(Clock)=\phi$ であるので、$C_B(Dlock)$ と $C_B(Clock)$ は比較することができない。

ここで、m_1 が他のロックモードよりも頻繁に使用されるとする。このとき、m_1 と競合する演算は実行の待ち状態が増える。ここで、$\psi(m)$ をロックモード m の使用頻度とする。使用頻度とは、一定期間内に o_i に発行する全てのロックモードに対し

て m を使用する演算の割合を示し、o_i にかけられるロックの頻度の和は、$\Sigma_{m \in Mi}\ \psi(m) = 1$ と表せる。$\|C_i(m)\|$ は、ロックモード m の重みといい m と競合するロックモドの集合の頻度を示しており、$\Sigma_{m' \in Ci(m)}\ \psi(m')\ (\leq 1)$ で求められる。

【定義】 M_i 内の任意のロックモード m_1、m_2 に対して、$m_1 \in C_i(m_2)$、かつ $m_2 \in C_i(m_1)$、かつ $\|C_i(m_1)\| \geq \|C_i(m_2)\|$ が成り立つとき、m_1 は、m_2 よりも強い（$m_1 \leq m_2$）と定義する。

ここで、オブジェクト o_i に対し、演算 op_1、op_2 は、各ロックモード m_1、m_2 の演算であるとする。次に、op_i と競合するロックモードで o_i がロックされているとき、ロックが開放されるまで、op_i が実行するのを待つ場合のブロック率を考える。$\|C_i(m_1)\| \geq \|C_i(m_2)\|$ が成り立つ場合、op_i は、op_2 よりも高い確率でブロックされる。

例1の場合、$C_f(wlock) \supset C_f(rlock)$ が成り立つので、$wlock \leq rlock$ が成り立つ。例 2 の場合、*Alock*、*Clock*、*Dlock*、*Wlock* がシステム全体で各々、10%、20%、40%、30%の頻度で発行されるとする。すなわち $\psi(Alock)$=0.1、$\psi(Clock)$=0.2、$\psi(Dlock)$=0.4、$\psi(Wlock)$=0.3 であったとする。ここで、$\|C_B(Clock)\| = \|C_B(Alock)\| = \psi(Dlock) + \psi(Wlock)$=0.7、$\|C_B(Dlock)\| = \|C_B(Wlock)\| = \psi(Alock)$=0.3 と表せる。また、$\|C_B(Dlock)\| < \|C_B(Clock)\|$、$\|C_B(Wlock)\| < \|C_B(Dlock)\|$、$\|C_B(Alock)\| \leq \|C_B(Clock)\|$ が成り立つので、$Dlock \leq Clock$、$Wlock \leq Clock$、$Alock \leq Clock$ が成り立つ。*Clock*は、*Alock*、*Dlock*、*Wlock* よりも高い確率でブロックされる。

M_i 内のロックモードは、重みの強さの関係“≤”で半順序付けられる。M_i 内に $m \leq m'$ の条件を満たすロック m' が存在しないときに限り、m は、M_i 内で極大である。M_i 内の全ての m' に対して、$m' \leq m$ の条件を満たす m が存在するときに限り、m は最大値である。同様に、極少、最小値が定義できる。M_i 内の任意の m_1、m_2 に対して、$m_1 \leq m$、$m_2 \leq m$、$m_1 \leq m' \leq m$、かつ $m_2 \leq m' \leq m$ の条件を満たす m' が存在しないならば、m は、最小上界（*lub*）$m_1 \cup m_2$ であるという。同様に、最大下界（*glb*）$m_1 \cup m_2$ が定義できる。

5.3.2 ロックプロトコル

既存のシステムは、2 種類の演算しかサポートしていない。すなわち *read*、*write*演算のみを考慮している。悲観的な *2PL* では、*write*演算を行う際に、全てのレプリカが最も強いロックモード *wlock* でロックされている。楽観的な *2PL* では、レプリカは初め、最も弱いロックモード *rlock* でロックされ、コミット時に *wlock* でロックされる。オブジェクトに基づくシステムにおいては、オブジェクトは、抽象的な演算をサポートしている。さらに、ロックモードの集合 M_i 内の全てのロックモードは、前節で議論した強さの関係“≤”によって半順序付けることができる。本調査研究では、オブジェクトに基づくロック方式（*OBL*：object-based locking）を議論する。本方式は初めにレプリカを $\mu(op_i)$ よりも弱いロックモード $\nu(op_i)$ でロックし、コミット時に $\mu(op_i)$ でロックする方式をとる。ここで、$\nu(op_i)$、$\mu(op_i)$ は、$\nu(op_i) \leq \mu(op_i)$ の条件を満たすものとする。$\nu(op_i)$ は、システム内で最も弱いロッ

クモードではなく、同様に、$\mu(op_i)$ は、システム内で最も強いロックモードではない。op_i の競合確率が高ければ、$\nu(op_i)$は、重みの上で強いロックモードとなる。さらに、レプリカの集合 R_i 内でロックされるレプリカの個数 $Q(op_i)$ は、op_i の重みの強さに基づいて決定される。*OBL* プロトコルは、楽観的な方式と悲観的な方式の間に位置付けられる。ここで、既存の read、write 演算に基づく方式を抽象演算に適用することを考えたとき、例えば、*Deposit*、*Withdraw* をサポートする銀行オブジェクト B を考える。*Deposit*、*Withdraw* は、銀行オブジェクトの状態を更新するので、コミット時に *wlock* でロックされる。ここで、トランザクションが、*Deposit*、*Withdraw* を 銀行オブジェクト B に発行したとき、他のトランザクションが *rlock* で B をロックしているならば、*rlock* から *wlock* に変換することができなくなり、*Deposit*、*Withdraw* はアボートする。このように既存の *read*、 に基づく方式では、多くのトランザクションがアボートしてしまう。しかし、オブジェクトに基づく方式では、*Deposit* と *Withdraw* は競合しない。すなわち、トランザクションがコミット時に、*Dlock*、または *Wlock* で B をロックされていたとしても、トランザクションはロックモードを *Dlock*、*Wlock* に変換することができる。このように、既存の read、write に基づく方式よりも、オブジェクトに基づく方式では、多くのトランザクションをコミットすることができる。

ここで、トランザクション T は o_i 上で演算 op_{it} を呼び出すと仮定する。初めに $R(o_i)$ 内の数個のレプリカを、$\mu(op_{it})$ よりも弱いロックモード $\nu(op_{it})$ でロックする。$q(op_{it})(\leq r_i)$ を op_{it} の実行前に、op_{it} によりロックされるレプリカ数とする。$q(op_{it})$ 個全てのレプリカがロックできないときは、op_{it} はアボートする。$q(op_{it})$ 個全てのレプリカがロックされたときは、ロックされたレプリカ上で op_{it} が実行される。Tがコミットするとき、任意の $Q(op_{it})$ 個のレプリカが、$\mu(op_{it})$ のロックモードでロックされる。ここで、$q(op_{it})$ と $Q(op_{it})$は、$q(op_{it}) \leq Q(op_{it})$の条件を満たし、$\nu(op_{it})$ と $\mu(op_{it})$ は$\nu(op_{it}) \leq \mu(op_{it})$ の条件を満たすこととする。

次に、ロックされるレプリカ数 $q(op_{it})$と $Q(op_{it})$の数と、ロックモード $\nu(op_{it})$ について議論する。システム内でレプリカのロック数が増加すると、演算のアボート率が高くなる。よって、op_{it} が呼び出される回数が増加するに従い、ロックされるレプリカ数を少なくする方式を提案する。$q(op_{it})$、$Q(op_{it})$ は、o_i のサポートする他の演算と op_{it} が競合する確率に基づき決定される。決定方法は、以下の q と Q の制約に従う。

ここで、r_iは、レプリカの総数である。

(1) $\mu(op_{it})$ が $\mu(op_{iu})$ と競合するならば、$Q(op_{it})+Q(op_{iu})>r_i$、かつ $Q(op_{it})+Q(op_{iu})>r_i$が成り立つ。

(2) $\|C_i(op_{it})\| \geq \|C_i(op_{iu})\|$ が成り立ち、かつそのときに限り、$Q(op_{it}) \geq Q(op_{iu})$が成り立つ。

ここで op_{it} が、$R(o_i)=\{o_i^1, ..., o_i^{ri}\}$ 内のレプリカに発行され、$Q(op_{it})$ 個のレプリカをロックするとする。また、互いに競合する op_{it}、op_{iu} が、o_i 上で呼び出されたと

する。 op_{it}、op_{iu} が共に、レプリカをロックできる確率は、$1-[Q(op_{it})\cdot\psi(op_{it})/r_i]\,[Q(op_{iu})\cdot\psi(op_{iu})/r_i]$ で求められる。ここで $C_i(op_{it})$ は、o_i 上で op_{it} と競合する演算の集合であるとする。op_{it} により $Q(op_{iu})$ 個のレプリカがロックされる確率 $P(op_{iu})$は、$1-\Pi_{opiu\in Ci(opit)}\,[Q(op_{it})\cdot\psi(op_{iu})/r_i]$ で求められ、$P(op_{it})$を最小値にするように $Q(op_{it})$は決定される。

本調査研究では、簡単化した OBL プロトコル示す。すなわち、$q(op_{it})$ と $Q(op_{it})$ を同数にし（$Q(op_{it})=q(op_{it})$）、ロックモードッ(op_{it})は、o_i がサポートする演算 op_{it} 全てに対して同じであることとする。o_i がサポートするレプリカの集合 R_i 内の全てのレプリカに対して基底のモードとして ⊥ を定義する。⊥ は、M_i 内の全てのロックモード m に対して、$\perp \le m$ が成立するロックモードとして定義される。⊥ は ⊥ と競合しないが、他の全てのロックモードと競合する性質を持っているものとする。ここで、$M_i=\{rlock, wlock\}$ であるならば、$rlock \le wlock$ であるので、⊥ は、$rlock$ となる。M_i 内に最小値があるならば、基底のモード ⊥ は M_i 内に含まれ、そうでなければ、システム内に ⊥ を用意する。

トランザクション T 内の演算 op_{it} が、o_i のレプリカ $\{o_i^1, ..., o_i^{ri}\}$ をロックするとき、o_i は以下のロック手順に従う。

【ロック手順】

(1) $q(op_{it})$個のレプリカを $R(o_i)$ 内からランダムに選出する。ここで、Q_{it}を$R(o_i)$ 内から選出されたレプリカの部分集合とする。

(2) Q_{it} 内の全てのレプリカは基底のモード ⊥ でロックされる。
Q_{it} 内の全てのレプリカのロックに成功した時、Q_{it} 内のレプリカは、op_{it} によって操作される。

(3) T がコミットする時、Q_{it} 内の全てのレプリカのロックモードを $\mu(op_{it})$ に変換し、変換が成功すれば op_{it} はコミットし、成功しなければアボートする。

5.3.3 Version vector

ロックされたレプリカ内で最新のレプリカを検出する方法として、Version vector を示す。ここで例として、4 つの演算 *Deposit*(*D*)、*Withdraw*(W)、*Check*(C)、*Audit*(A) をサポートしている銀行オブジェクト B を考える。銀行オブジェクト B は、4 つのレプリカ B^1、B^2、B^3、B^4 を持っているものとする。B 内の各演算の競合関係は［図5.1］のように D、W がそれぞれ C、A と競合する関係であるとする。

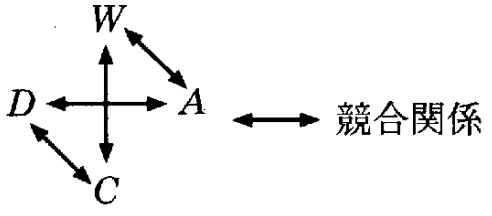


図5.1：競合関係

D、W、Aは、unstable 演算である。ここで、Q_{it}を演算 op_{it} のコーラムとする。コーラムは、O_i 上で op_{it} が実行する際にロックされるレプリカの総数であるとする。コータリーの理論に基づくならば、O_i 上で、op_{it} と op_{iu} が競合するとき、$Q_{it}+Q_{iu}>r_i$ が成立しなければならない。各コーラムの数は、$Q_D=2$、$Q_W=2$、$Q_C=3$、$Q_A=3$、とする。各々のレプリカ B_i は、初期値 0 の version 番号 $V(B^i)$ を持つ。

ここで、演算 D が二つのレプリカ B^1、B^2 に発行され、その後、演算 W が B^3、B^4 に発行されることを考える。このとき、全てのレプリカの version 番号は 1 に変わる。すなわち、$V(B^1)=V(B^2)=V(B^4)=V(B^4)=1$ となる。次に、演算 C が B^1、B^2、B^3 に発行されたときを考える。ここで B^3 の状態は B^1、B^2 の状態と異なっている。Cは 1 つのレプリカ上で、例えば B^1 で実行される。ここで、D と Wの実行後、各レプリカの version 番号が全て 1 であることが問題となる。すなわち、システム内で最新レプリカを捜し出すことができない。しかし、D と W は競合しないため、D、W は実行順序によりオブジェクトの状態が異なることはない。したがって、B^1、B^2 で実行された D が B^3、B^4 で実行され、かつ B^3、B^4 で実行された W が B^1、B^2 で実行されることにより、全てのレプリカの内容を同じにすることができる。

既存の方式に基づけば、全ての unstable 演算 D、W、A は、write 演算と考えられる。Cのような stable 演算は read 演算として考えられる。すなわち、$Q_D+Q_W>4$ を満たさなければならない。例えば、$Q_D=3$、かつ $Q_W=3$ とする。すなわち、Dは、B^1、B^2、B^3 に発行され、$V(B^1)=V(B^2)=V(B^3)=1$ となる。このとき、B^1、B^2、B^4 に W が発行されるとする。ここで、$V(B^1)=V(B^2)(=1)>V(B^4)(=0)$であるので、$W$ は B^1、B^2でランダムに選出されたレプリカ、例えば B_1で実行され、$V(B^1)=2$ となる。B^2、B^4 は B^1 の内容を用いて更新される。ここで、$V(B^1)=V(B^2)=V(B^4)=2$、$V(B^1)=1$ となる。我々が、抽象演算間の競合関係に基づきコーラムを決定するのであれば、ロックされるレプリカ数を減らすことができる。しかしロックされるレプリカ数を減らせば、Version 番号を使って、最新のレプリカを選ぶことは困難である。

本調査研究では、ロックされたレプリカの集合内で最新のレプリカを検出するために*Version vector*を提案する。各々のレプリカo_i^h は Version vector $V_{it}^h=\langle V_{il}^h, ..., V_{ili}^h\rangle$ を持つとする。各々の要素 V_{it}^h は、演算 op_{it} に対しての o_i^h の version 番号を示している。V_{it}^h は $\langle v_{it}^h, b_{it}^h\rangle$ の組合せで表される。v_{it}^h は op_{it} に対しての o_i^h の version 番号を示している。op_{it} が o_i^h で実行され、かつ op_{it} が unstable 演算であるならば、演算実行ごとに、v_{it}^h は+1 される。b_{it}^h は、どのレプリカに op_{it} が発行されたかを示すビット列であり $(b_{it}^h, ..., b_{it}^{ri})$ で表される。op_{it} が o_i^k に発行されたならば、o_{it}^kは 1になる。発行されなければ、0 のままである。例えば、レプリカB^i の Version vector はa、$\langle V_{BD}^i, V_{BW}^i, V_{BC}^i, V_{BA}^i\rangle$ で表される。Version vector の初期状態は、$V_B^i=\langle 0_{0000}, 0_{0000}, 0_{0000}, 0_{0000},\rangle$ と表される。ここで、$v_{b1\,b2\,b3\,b4}$は、Version vector の要素を示しているとする。つまり、$\langle$v, (b^1、 b^2、 b^3、 b^4)$\rangle$ と表せる。さらに、各レプリカは演算の実行履歴を持っているものとする。

ここで D が B^1、B^2 に発行されたならば B^1、B^2 の Version Vector は、$V_B^1 = V_B^2 = \langle 1_{1100}, 0_{0000}, 0_{0000}, 0_{0000}, \rangle$ と表せる。そのとき、W が B^3、B^4 に発行されたならば、$V_B^3 = V_B^4 = \langle 0_{0000}, 1_{0011}, 0_{0000}, 0_{0000}, \rangle$ となる。さらにここで、C が、B^1、B^2、B^3 に発行されたならば、各々の Version vector には、$V_B^1 = V_B^2 \neq V_B^3$ の関係がある。ここで、$V_{BD}^1 = V_{BD}^2 = 1_{1100}$、かつ $V_{BW}^3 = 1_{1100}$ であるので、B^1、B^2 に D が発行され、B^3 に W が発行されていることがわかる。さらにここでは最新のレプリカがないことがわかる。つまり、D、W、2 つの演算が実行されているレプリカがない。この場合、ロックされた各レプリカの Version vector の要素をレプリカ間で比較し、最も新しいVersion vecotor の要素を選出する。D の演算に対応する最新の Version vector の要素は B^1 と B^2 が持っており、ランダムにどちらかのレプリカが選出される。ここでは B^1 が選出されたとする。W の演算に対応する最新の Version vector の要素は B^3 が持っている。ここで、各演算に対する最新の Version vector の要素を持っているのが B^1 と B^3 ということになる。次に一つのレプリカ、例えば B^1 が選出され、B^3 で実行された演算 W が B^1 で実行される。すなわち B^1 のレプリカのVersion vectorは、$V_B^1 = \langle 1_{1100}, 1_{0011}, 0_{0000}, 0_{0000}, \rangle$ となる。ここで、C は B^1 で実行される。 C は stable 演算であるので、V_{BC}^1 は変化しない。[図5.2]。

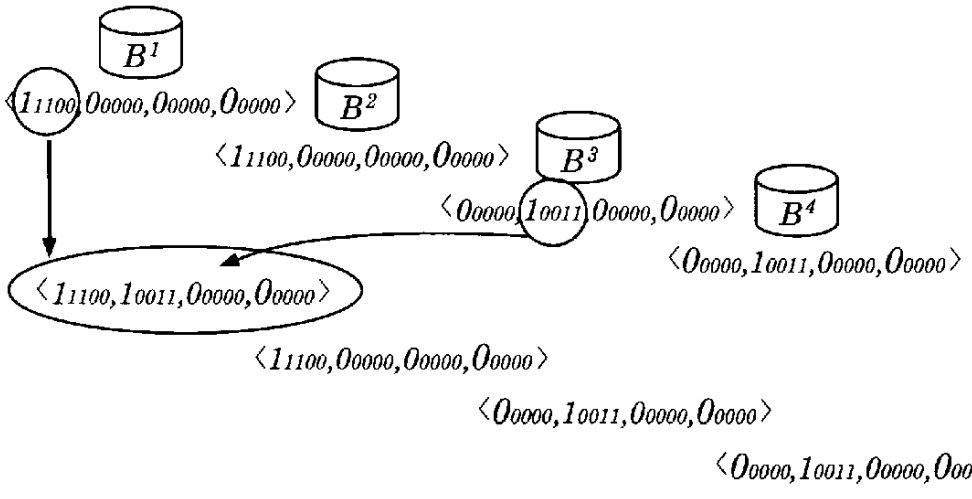


図5.2：演算の受渡し

次に、D が B^3、B^4 に発行されたならば、各々の Version vector は、$V_B^3 = V_B^4 = \langle 1_{0011}, 1_{0011}, 0, 0 \rangle$ となる。ここで、A が、B^2、B^3、B^4 に発行されたならば、B^2 の Version vector の D の要素 は $V_{BD}^2 = 1_{1100}$であり、B^3 の Version vector の D の要素 は $V_{BD}^3 = 1_{0011}$であるので、内容の異なる演算 D が B^2 と B^3 に発行されたことがわかる。さらに、B^2 の Version vector の W の要素 は $V_{BW}^2 = 0_{0000}$であり、B^3 の Version vector の W の要素 は $V_{BW}^3 = 1_{0011}$ である。よって、A の演算が発行されたレプリカ内には、最新のレプリカがないことがわかる。この場合、ロックされた各レプリカのVersion vector の要素をレプリカ間で比較し、最も新しい Version vecotor の要素を選出する。D の要素で最新のレプリカは 2 つあり、1_{1100}の D の演算に対応する最新のVersion

vectorの要素は B^2 が持っており、1_{0011}の D の演算に対応する最新のVersion vectorの要素は B^3、B^4 が持っており、 Wの演算に対応する最新のVersion vectorの要素は B^3、B^4 が持っている。ここで、各レプリカは最新のVersion vectorの要素をもつレプリカの演算の実行履歴を参照し、各レプリカでまだ実行されていない演算を実行する。まず、B^3 で実行された D、W の演算を B^2 で実行する。ここで D と W は競合しないので、B^2 での実行順序が変わっても結果は変わらない。また B^2 で実行された D の演算を B^3、B^3 で実行する。この後 A は B^2、B^3、B^4 で実行される。この結果、B^2 のVersion vector は、$V_B^2 = \langle 1_{1100}, 1_{0011}, 0_{0000}, 1_{0111} \rangle$ となり、B^3、B^4 の Version vecotor は、$V_B^3 = V_B^4 = \langle 1_{1100}, 1_{0011}, 0_{0000}, 1_{0111} \rangle$ となる [図5.3]。

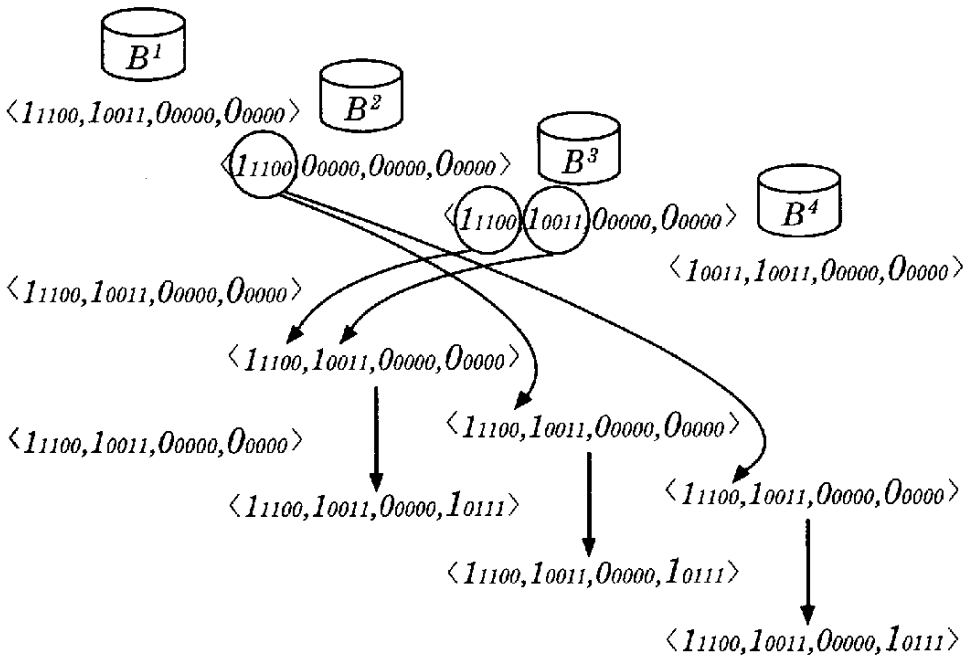


図5.3：演算の受渡し

Version vector 間の関係について議論する。b^h と b^k をそれぞれ o_i^h と o_i^k のビット列とする。全ての j に対して、$b^{kj} = 1$、$b^{hj} = 1$ であるとき、b^h は、b^k に含まれる（$b^h \subseteq b^k$）こととする。$b^h \cup b^k$ は、$\langle b^1, ..., b^n \rangle$ を示している。ここで、$b^{hj} = b^{kj} = 1$ ならば $b^j = 1$ であり、そうでなければ $b^j = 0$ である。

(1) $v_{it}^h \leq v_{it}^k$、かつ $b_{it}^h \subseteq b_{it}^k$が成立するならば、$V_{it}^h \leq V_{it}^k$が成立する。

(2) 任意の演算 op_{it} に 対して$V_{it}^h \leq V_{it}^k$が成立するならば、$V_i^h \leq V_i^k$ が成立する。

$b_{it}^h \cap b_{it}^k \neq \phi$ であるならば、例えば、$v_{it}^h \leq v_{it}^k$であったとしても V_i^hとV_i^kには順序関係はない。例えば、$V_B^1 = \langle 1_{1100}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$ であり、 $V_B^3 = \langle 2_{1110}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$ であるとき、$V_B^1 \leq V_B^3$が成立する。しかし、$V_B^2 = \langle 2_{0011}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$ であるならば、V_B^1と V_B^2は比較することはできない。o_iのレプリカの集合N ($N \subseteq R$ (o_i)) 内に、$V_i^k \leq V_i^h$ の関係を満たすような、 o_i^kの Version vector V_i^kがないときは、V_i^hは、最大であるという。

以下に Version Vector の要素 V_{it}^h と V_{it}^kの演算 ∪ を定義する。

●$V_{it}^h \cup V_{it}^k =$

$$\begin{cases} V_{it}^h \leq V_{it}^k \text{ならば、} V_{it}^k \\ V_{it}^h \leq V_{it}^k \text{ならば、} V_{it}^h \\ \text{上記以外の場合} \langle max\ (v_{it}^h, v_{it}^k)+1,\ b_{it}^h \cup b_{it}^k \rangle \end{cases}$$

●$V_i^h \geq V_i^k = \langle V_{i1}^h \cup V_{i1}^k, ..., V_{il}^h \cup V_{ili}^k \rangle$.

例えば、$V_B^2 = \langle 1_{0011}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$ そして、$V_B^3 = \langle 2_{0011}, 1_{0011}, 0_{0000}, 0_{0000} \rangle$ ならば、$V_B^2 \cup V_B^3 = \langle 2_{0011}, 1_{0011}, 0_{0000}, 0_{0000} \rangle$ が成立する。

【定義】 任意の競合しない演算の組 op_{it} と op_{iu} に対して競合する、任意の unstable 演算 op_{iv} が、$V_{iv}^h = V_{iv}^k$の関係を満たす時、V_i^h は V_i^kと等価（$V_{iv}^h \equiv V_{iv}^k$）である。

例えば、$V_B^1 = \langle 1_{1100}, 0_{0000}, 0_{0000}, 2_{0101} \rangle$ は、$V_B^2 = \langle 0_{0000}, 1_{0011}, 0_{0000}, 2_{0101} \rangle$ と等価である。V_i^h と V_i^k が等価であるならば、o_i^h と o_i^k は、レプリカ上で、まだ実行されていない競合しない演算を実行することにより、同じ状態になる。例えば、B^1上で、Wが実行され、かつ B^2 上で D が実行されたならば、B^1 と B^2 を同じ状態にすることができる。

Version vector を用いることによるロックプロトコルを議論する。ここで、op_{it} がオブジェクト o_iに発行されるとする。o_{it}は、op_{it}のコーラムを表すとする。N_{it}をop_{it}によりロックされるレプリカの部分集合であるとする。$|N_{it}|$（N_{it} 内のレプリカの総数）$= Q_{it}$であり、$N_{it} \subseteq R\ (o_i)$ である。o_i^hは、任意の演算 op_{it}に対して演算の実行履歴 l_{it}^hを持ち、o_i^h上で実行された op_{it} を l_{it}^hに取っておくものとする。

【ロックプロトコル】

任意のオブジェクト o_sは、N_{it} 内のレプリカに演算 op_{it} を発行する。

(1) N_{it} 内の全てのレプリカは、$\mu(op_{it})$ のロックモードでロックされ、レプリカのロックに成功しなければ、op_{it} はアボートする。N_{it} 内の各レプリカは、Version vector V_i^h と共にロックの有無の返答を o_s に返す。

(2) N_{it} 内の全てのレプリカから返信を受け、o_s は N_{it} 内の全てのレプリカの Version vector V_i^h を得る。次に o_sは、$V_i^h = V = \cup \{V_i^k \mid o_i^k \in N_{it}\}$ の条件を満たす最新のレプリカ o_i^h を 1 つ、N_{it} 内から選出する。条件を満たすレプリカがない場合、N_{it} 内の Version vector の要素ごとに最新のレプリカ o_i^h を全て選出する。

(3) o_i^h で実行されておらず、他のレプリカでは実行された演算 op_{iu} がシステム内にある場合、op_{iu} と op_{it} が競合するならば、op_{it} を実行する前に op_{iu} を o_i^h 上で実行する必要がある。ここで p_{it} を、o_i^h 上では実行されておらず、システム内の他のレプリカで実行された演算であり、かつ op_{it} と競合する演算 op_{iu} の系列であるとする。すなわち、$b_{iu}^h \cap b_{iu}^h = \phi$、かつ $v_{iu} \neq 0$ の条件を満たす Version vector の要素に対応する演算の系列であるとする。ここで、o_i^h は p_{it}内の演算を実行する。

(4) o_i^h の Version vector が $V_i^h = V$ になったならば、op_{it} は o_i^h 上で実行され、

$V_{it}^h = V_{it}^h + 1$ とする。

(5) op_{it} が unstable 演算であるならば、o_i^h で実行された演算を N_{it} 内の o_i^h 以外のレプリカでも実行する必要がある。ここで p_{it} を、o_i^h 上では実行され、o_i^k 上では実行されていない演算であり、かつ o_i^h で実行された演算の系列と競合する演算 op_{iu} の系列であるとする。o_i^k は op_{it} 内の演算を実行後、全てのレプリカ上で op_{it} を実行する。

(6) Version vector の要素は、$V_{it}^h = \langle V_{it}^h, V_{it}^h \rangle$ で表され、Version vector の要素が仮に、$b_{it}^h = \langle 1, ..., 1 \rangle$ になったならば、o_s は op_{it} が実行されていない全てのレプリカに対して、op_{it} を発行し、o_i^k は (5) の手順に従い op_{it} を実行する。このとき $j = 1, ..., r_i$ に対して、$b_{it}^{hj} := 0$ とし、l_{it}^h 内の全ての演算 op_{it} は消去される。

(7) o_i^h は、op_{it} の演算結果を o_s を返す。全てのビット列 b_{it}^{hj} が 1 になるとき、すなわち、$b_{it}^h = \langle 1, ..., 1 \rangle$ になるとき、version 番号 V_{it}^h と ビット列 b_{it}^h は初期化される。すなわち、$v_{it}^h := 0$ 、 $b_{it}^h := \langle 0, ..., 0 \rangle$ となる。このようにして、v_{it}^h は、o_i^t 上でいくつのインスタンスが実行されたかを示している。

【定義】 任意の unstable 演算 op_{it} に対して、$b_{it}^h \cap b_{it}^k \neq \phi$ が成り立つならば、$b_{it}^h = b_{it}^k$、$b_{it}^h = b_{it}^k$ が成り立つ。

unstable 演算 op_{it} に対して、$b_{it}^h \cap b_{it}^k = \phi$、かつ $v_{it}^h > 0$、または $v_{it}^k > 0$ が成り立つとき、o_i^h 上で実行された op_{it} の実行系列 s^k と、o_i^k 上で実行された op_{it} の実行系列 s^k は異なる。s^h と s^k は、各々 v_{it}^h、v_{it}^k と 演算の実行履歴を持っており、o_i^h と o_i^k の一貫性を保つために、s^k と s^h は o_i^h と o_i^k 上で各々実行される。

6. まとめ

今後の情報システムでは、システムの構成の変化、負荷等の変化に適応できるとともに、利用者、アプリケーションの要求の変化に対して、柔軟に適応できることが求められてきている。本調査研究では、分散システムは、CORBA 等で議論されているように、複数の様々なオブジェクトから構成されているものとして考察を行なった。複数のオブジェクトから構成される分散システムを、どのようにやわらかなシステムとしていくかについて論じた。オブジェクトの提供するサービスを、サービスの型（種類）とサービス品質（QoS）に分けて考察を行なった。特に、サービス品質（QoS）に着目し、要求される QoS をサポートする方法を示した。

利用者から見て、オブジェクトは、データ構造とこれに対する操作演算（メソッド）を提供するブラックボックスである。オブジェクトが提供するサービスは、演算を用いて得ることができる。ここで、サービスを、種類（type）と品質（quality）の二つの側面に分けて考える。サービス種類は、演算の種類を示している。各サービスは、応答時間、画像の品質等のパラメータが異なっている。このパラメータは、サービス品質（QoS：Quality of Servicxe）と呼ばれる。ここでは、特に、ビデオ、イメージ、音声等のマルチメディアオブジェクトを例として考え、オブジェクトのサービス品質について検討を行った。オブジェクトがシステム環境の変化により、オブジェクトの提供するサービスが変化する。本調査研究では、オブジェクトの変化をオブジェクトの提供する『QoSの変化』としてモデル化する。

オブジェクトは、データ構造とこれに対する操作演算（メソッド）が一体化したものである。利用者要求の変化とシステム環境の変化に対して、自律的に適応する機構を考える必要がある。本調査研究では、オブジェクトの提供する QoS に着目して、以下についての検討を行った。

(1) QoS に基づいた演算間の関係

(2) QoS に基づいたオブジェクトの多重化方法

これまでのシステムでは、オブジェクトのある状態に二つの演算 op1 と op2 を適用したときに、同じ状態が得られるならば、op1 と op2 は同値である。ここでは、QoS に着目して、得られた状態が異なっていても、二つの状態が同じ QoS を提供するならば同値と考える。例えば、80fps（frame／sec）の画像を記憶しているオブジェクトがあるとする。このオブジェクトが、80fps で表示できる演算 display1 と、40fpsでしか表示できない演算 display2 を持っているとする。このとき、アプリケーションが 30fps での表示を期待しているのであれば、どちらの演算を適用してもよい。このように、新しく QoS を考えた演算間の関係を検討した。

この QoS に基づいた演算間の関係を基にして、オブジェクトの多重化についての検討を行った。これまでのシステムでは、オブジェクトのレプリカは、全く同じデータと演算を持っているものとして考えられてきていた。これに対して、新しくオブジ

ェクトの提供する QoS について考え、アプリケーションの提供する QoS を提供しているオブジェクトをレプリカとする概念を示した。例えば、あるオブジェクトがアプリケーションの要求するQoSを提供できなくなったとき、全く同じオブジェクトでなくても要求する QoS を提供しているオブジェクトによりサービスを提供することができる。オブジェクトのレプリカ間の一貫性を保つための方法として、新たに OBL (object-based locking) 方式を提案した。

このように、性能の劣化等のシステム環境の変化とアプリケーションの要求の変化を QoS の変化と考えることにより、これらの変化に容易に適応できる機構を示すことができた。これらの機構をエージェントに組み込むことにより、やわらかい分散システムとすることができる。

参考文献

[1] Barbara, D. and Imielinski, T., "Sleepers and Workaholics : Caching Strategies in Mobile Environments," *Proc. of the ACM SIGMOD*, pp. 1—12, 1994.

[2] Bernstein, P. A., Hadzilacos, V., and Goodman, N., "Concurrency Control and Recovery in Database Systems," *Addison-Wesley*, 1987.

[3] Carey, J. M. and Livny, M., "Conflict Detection Tradeoffs for Replicated Data," *ACM TODS*, Vol. 16, No.4, pp. 703—746, 1991.

[4] Ellis, C. A., Gibbs, S. J., and Rein, G. L., "Groupware," *Comm. ACM*, Vol.34, No.1, pp.38—58, 1991.

[5] Gruber, R., Kaashoek, F., Liskov, B., and Shrira, L., "Disconnected Operationin the Thor Object-Oriented Database System," *Proc. of IEEE Workshop on Mobile Computing Systems and Applications*, pp. 51—56, 1994.

[6] Jing, J., Bukhres, O., and Elmagarmid, A., "Distributed Lock Management for Mobile Transactions," *Proc of IEEE ICDCS*—15, pp. 118—125, 1995.

[7] Kistler, J. J. and Satyanaranyanan, M., "Disconnected Operation in the Coda File System," *ACM Trans. on Database Systems*, Vol. 10, No. 1, pp. 2—25, 1992.

[8] Kung, H. T. and Robinson, J. T., "On Optimistic Methods for Concurrency Control," *ACM Trans. on Database Systems*, Vol.6, No.2, pp. 81—87, 1994.

[9] Moss, J. E., "Nested Transactions : An Approach to Reliable Distributed Computing," *The MIT Press Series in Information Systems*, 1985.

[10] Silvano, M. and Douglas C, S., "Constructing Reliable Distributed Communication Systems with CORBA" *IEEE Communications Magazin*, Vol.35, No.2, pp.56—60, 1997.

[11] Yoshida} Yoshida, T. and Takizawa, M., "Model of Mobile Objects," Proc. of the 7th DEXA(*Lecture Notes in Computer Science, No.1134, Springer-Verlag*), pp. 623—632, 1996.

[12] Bernstein, P. A., Hadzilacos, V., and Goodman, N., "Concurrency Control and Recovery in Database Systems," *Addison-Wesley Publishing Company*, 1987.

[13] Birman, K., Kenneth, P., and Renesse, V. R., "Reliable Distributed Computing with the Isis Toolkit," *IEEE Comp. Society Press*, 1994.

[14] Budhiraja, N., Marzullo, K., Schneider, B. F., and Toueg, S., "The Primary-Backup Approach," *Distributed Computing Systems, ACM Press*, 1994, pp.199—221.

[15] Cambell, A., Coulson, G., Garcfa, F., Hutchison, D., and Leopold, H., "Integrated Quality of Service for Multimedia Communication," *IEEE InfoCom*, 1993, pp.732—793.

[16] Campbell, A., Coulson, G., and Hutchison, D., "A Quality of Service Architecture," *ACM SIGCOMM Computer Communication Review*, Vol. 24, 1994, pp.6—27.

[17] Chandy, K. M., Misra, J., and Haas, L.˜M., "Distributed Deadlock Detection," *ACM TODS*, Vol.1, No.2, 1983, pp.144—156.

[18] Gall, D., "MPEG: A Video Compression Standard for Multimedia Applications," *Comm. ACM*, Vol.34, No.4, 1991, pp.46—58.

[19] Garcia-Molina, H. and Salem, K., "Sagas," Proc. of the ACM SIGMOD, 1987, pp.249—259.

[20] Higaki, H. and Hirakawa, Y., "Group Communication for Upgrading Distributed Programs," Proc. of the 16th IEEE International Conference on Distributed Computing Systems (ICDCS), 1996, pp.420—427.

[21] Higaki, H. and Takizawa, M., "Group Communication Protocol for Flexible Distributed Systems," *Proc. of the IEEE 4th Int'l Conf. on Netwok Protocols (ICNP-96)*, 1996, pp.48—55.

[22] Korth, H. F., Levy, E., and Silberschalz, A., "A Formal Approach to Recovery by Compensating transactions," *Proc. of the VLDB*, 1990, pp.95—106.

[23] Sabata, B., Chatterjee, S., Davis, M., and Syidir, J. J., "Taxonomy for QoS Specifications," *Proc of the IEEE Computer Society 3rd International Workshop*

on Object-oriented Real-time Dependable Systems (*WORDS* '97), 1997, pp.100—107.

[24] Schmidt, D. C., Gokhale, A. S., Harrison, T. H., and Parulkar, G., "A High-Performance End System Architecture for Real-Time CORBA," *IEEE Communications Magazine*, 1997, pp.72—77.

[25] Schneider, B. F., "Replication Management using the State-Machine Approach," *Distributed Computing System, ACM Press*, 1993, pp.169—197.

[26] Takizawa, M. and Yasuzawa, S., "Uncompensatable Deadlock in Distributed Object-Oriented Systems," *Proc. of IEEE Int'l Conf. on Parallel and Distributed Systems* (*ICPADS*-92), 1992, pp.150—157.

[27] Tanaka, K. and Takizawa, M., "Distributed Checkpointing Based on Influential Messages," *Proc. of the 4th IEEE Int'l Conf. on Parallel and Distributed Systems* (*ICPADS*-96), 1996, pp. 440—447.

[28] Yoshida, T. and Takizawa, M., "Model of Mobile Objects," *Proc. of the 7th Int'l Conf. on Database and Expert Systems Applications* (*DEXA* '96), 1996, pp. 623—632.

平成10年３月発行

発　行　　財団法人　データーベース振興センター
東京都港区新橋二丁目１３番８号
新橋東和ビル５階
ＴＥＬ０３－３５０８－２４３０

委託先　　株式会社　シネ・ジャーナルプロダクション
東京都新宿区四谷４丁目２１番３７号
ＴＥＬ０３－３３５５－１５９４

印刷所　　株式会社　現代工学社
東京都新宿区四谷２丁目１３番
ＴＥＬ０３－３３５７－７１６１