０８－開－１２

データベース構築促進及び技術開発に関する報告書

インターネット上の情報オブジェクトを利用した高信頼アプリケーション開発技術に関する調査研究

平成９年３月

財団法人　データベース振興センター

委託先　株式会社新世代システムセンター

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「インターネット上の情報オブジェクトを利用した高信頼アプリケーション開発技術に関する調査研究」は平成８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、当財団が株式会社新世代システムセンター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８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９年３月

財団法人　データベース振興センター

平成８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　報道写真を中心とした商用デジタル写真データベース構築	㈱毎日新聞社
	2　WWWによる医薬情報全文検索データベースの構築と利用者Q＆Aデータベースシステムの構築	日本電子計算㈱
	3　生活博物史データベースの構築	㈱ＮＨＫ情報ネットワーク
中小企業振興 地域活性化	4　信濃毎日新聞記事データベースの構築	信濃毎日新聞㈱
	5　患者投薬時に交付する服薬等指導文書のデータベース構築	㈱ＹＳ企画
	6　自治体議事録のＳＧＭＬデータベース化と情報検索ブラウザ機能開発	㈱会議録研究所
	7　全国ベンチャー支援機関のネットワーク化による起業化支援データベースのプロトタイプ構築	㈱日本インテリジェントトラスト
	8　インターネットを用いたイベント情報サービス	（社）日本イベント産業振興協会
技　術	9　データベースクリアリングサーバのプロトタイプの作成	セントラル開発㈱
	10　中堅・中小企業向け顧客データベース利用ソフトのプロトタイプ作成	情報図書館ＲＵＫＩＴ ㈱日経リサーチ
	11　写真データベースへの感性からの接近に関する調査研究	㈱中日新聞社
	12　インターネット上の情報オブジェクトを利用した高信頼アプリケーション開発技術に関する調査研究	㈱新世代システムセンター

目　　次

Ⅰ．課題性の研究

1．目的

インターネットの普及により、リレーショナルデータベースシステムを始めとして、WWWサーバ等の種々の情報システムが相互接続されてきている。一方、分散情報システムの枠組みとして、CORBA等のオブジェクト指向システムが提案され、利用されている。ここでは、こうした情報システムをオブジェクトとしてとらえることにする。現在のところ、インターネット上のフリーウェア、シェアウェア等のオブジェクトは、利用者の要求に十分応える信頼性、可用性、安全性を満たしていないのが実状である。これからの情報システムでは、インターネット上のこうした膨大な、しかも低信頼なオブジェクトを利用して、利用者の要求に応えていくことが求められる。

このため、本調査研究では、インターネット上の種々のオブジェクトに対して、信頼性と可用性を提供する機構について研究する。本調査研究の実施により、インターネット上のデータ資源を有効に利用するための方法について明らかにし、わが国のデータベースの振興に資することを目的とする。

2．実施内容

本調査・研究における主な実施内容は、以下の項目に示す通りである。

(1) インターネット上の情報資源の調査

フリーウェア、シェアウェア等インターネット上の情報資源に関する調査。

(2) オブジェクト指向システム

オブジェクトを総合的に扱うためのモデルの研究。

(3) オブジェクトの信頼性、可用性技術

低信頼なオブジェクトを用いて、利用者が要求する品質（信頼性、可用性）提供する方法についての研究。

(4) オブジェクトの安全性

オブジェクト間での情報の流れを制御することによる安全性の提供についての方策の研究。

(5) 今後の課題

インターネットの急速の進展に伴う、より一層要求されるオブジェクトの品質保証についての今後の課題。

3. 実施体制

株式会社新世代システムセンター内に当該事業遂行のための「特別調査・研究委員会」を設置し、事業の実施に当った。

調査研究委員会

滝沢　誠（東京電機大学理工学部・経営工学科教授）

市川　隆（日本情報処理開発協会・常務理事）

山鳥雄嗣（日本情報処理開発協会・調査部長）

道田国雄（イフ・アドバタイジング調査部長）

尾崎春雄（シネ・ジャーナルプロダクション企画部長）

Ⅱ．インターネット上の情報オブジェクトを利用した高信頼アプリケーション開発技術

1章　はじめに

64 ビットの CPU を備えたワークステーションが登場し、コンピュータの高速化、高機能化、小型化が進んでいる。さらに、インターネットの普及により、世界規模でのネットワーク作りが進められている。無線を用いたモバイル通信の普及により、いつでもどこでもコンピュータを利用できる環境が整ってきている。こうしたコンピュータのインターネットを中心としたネットワーク化の普及により、リレーショナルデータベースシステム [CODD70, TAKI96] を始めとして、WWW サーバ等の種々の情報システムが相互接続されてきている。インターネットを用いて、世界中に分散した種々の情報を利用できるようになってきている。一方、コンピュータのオペレーティングシステム (OS) として Windows 95, NT, Unix が広く普及し、通信ネットワークのプロトコルとしては TCP/IP が実質的な国際標準として広く用いられてきている。これらの OS とネットワークを用いて、インターネットが構築され、情報システムの基盤技術が形成されている。一方、インターネット、Windows 等の標準 OS を用いたミドルウェアとして何が採用されるかが現在の重要な課題となってきている。

つまり、分散情報システムの共通の枠組みが必要となってきている。このため、OMG (Object Management Group) で開発されている CORBA (Common Object Request Broker Architecture) [MOWB95] 等のオブジェクト指向システム [KIM90] が提案され、利用されてきている。CORBA のインタフェース機能を備えたシステムが、Sun, Microsoft 等で開発されてきている。国内では、オブジェクト指向システムの OZ++ が開発されてきている。

インターネットの普及により、利用者は、世界中の情報資源を自由に獲得し、利用できるようになってきている。情報システムを設計、実現し、運用を行っていく場合に、新しくシステムを開発するのではなく、出来合いのシステムを利用することで、設計、実現、保守の時間 (ライフサイクル) を短縮することが求められている。ここでは、情報システ

ムを構成する種々の要素システムをオブジェクトとしてとらえることにする。インターネット上のフリーウェア、シェアウェア等のオブジェクトは、利用者が要求に応える十分な信頼性、可用性、安全性を提供していない。これからの情報システムでは、インターネット上のこうした膨大な、しかし低信頼なオブジェクトを利用して、利用者の要求に応えていくことが求められる。このために、本調査研究では、インターネット上の種々のオブジェクトに対して、信頼性と可用性を提供する機構について研究する。

本調査研究では、インターネット上のデータ資源を有効に利用するための方法について研究し、もってわが国のデータベースの振興に資することを目的とする。

(1) インターネット上の情報資源の調査

今後の情報システムの通信インフラとして急速に発展しているインターネットは、安価なフリーウェア、シェアウェア等の豊富な資源を自由に利用できる特長がある。したがって今後の情報システムは、こうしたインターネット上の情報資源を組み上げて、構築されていく傾向が強い。ここでは、インターネット上の情報資源としてどのようなものがあるかについての調査を行う。

(2) オブジェクト指向システム

インターネット上の膨大な情報資源を、統合的に捉えるモデルが必要となる。ここでは、オブジェクト指向システムをこのモデルとする。すでに、CORBAが大規模な情報システムのアーキテクチャとして国内外で利用され、国内では電総研で開発されているOZ++がある。このため、これらを基にしてモデルの研究を行う。

(3) オブジェクトの信頼性、可用性技術

インターネット上のオブジェクトは、一般に、信頼性、可用性についての保証がなされていない。こうした低信頼なオブジェクトを用いて、利用者が要求するサービス品質(信頼性、可用性)を提供することが必要となる。フリーウェア等のオブジェクトの持つバグに対しても、対処できなければならない。ここでは、このための技術として、オブジェクトを冗長配置することと、これらの間でチェックポイントを取ることにより、障害が起きた場合に過去の正しい状態に復旧する方法について研究する。

(4) オブジェクトの安全性

インターネット上のオブジェクトの安全性(security)をどのように提供するかは、EC (Electronic Commerse) 等で重要な課題となっている。ここでは、暗号化等のネッ

トワークの安全性ではなく、オブジェクトに対するアクセス制御について研究する。大規模な情報システムでは、あるオブジェクトAの情報が他のオブジェクトBに流出することにより、Aにアクセスできない利用者もBにアクセスすることにより、Aの情報にアクセスできることになる。ここでは、こうしたオブジェクト間での情報の流れ(フロー)を制御する方式について研究する。

本報告では、まず、第2章で、オブジェクト指向システムとCORBAについて述べる。第3章では、情報資源の調査と解析を行う。第4章では、分散システムの高信頼化、高可用化の技術について考える。第5章では、分散システムの安全性について述べる。

2章　オブジェクト指向システム

分散システムでは、ファイル、ディスク等のハードウェア資源から、ワープロ、データベースシステム、メール等の種々の資源が、複数のコンピュータに分散している。これらの各資源は、オブジェクトとして統合的にとらえられており、それぞれの資源は、一般に状態を持ち、状態に基づいてある操作を行っている。例えば、ファイルシステムは、ファイルに対して read, write 等の操作を行うシステムである。こうした資源を抽象化した概念がオブジェクト (object) である。オブジェクトを基本要素としたシステムとして、オブジェクト指向 (object-oriented) システムがある。

2.1　基本概念

まず、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。

(1) オブジェクト

(2) メッセージパッシング

(3) 階層構造(汎化、集約化)

(4) 継承

(5) ポリモルフィズム

これらについて、考える。

2.1.1　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) ）

メソッドとは、オブジェクトが提供する操作演算である。利用者は、メソッドを用いてのみオブジェクトを操作できる。オブジェクトのデータ構造をメソッド以外の演算を用いて

操作することはできない。この意味で、オブジェクトは、メソッドを抽象演算とした抽象データ型(ADT : Abstract Data Type)である。例として、オブジェクト『銀行』は、メソッドとして、入金、出金、照会といったものを提供している[図2.1]。利用者は、銀行オブジェクトに対して、メソッド入金を用いて、ある口座に入金を行える。

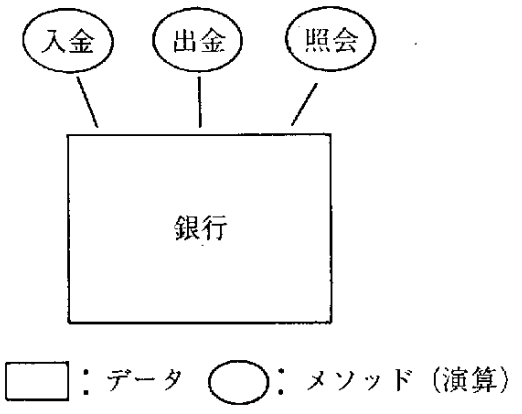


図2.1　オブジェクトの例

2.1.2　メッセージパッシング

オブジェクト間では、メッセージという通信単位の送受信により情報の交換が行われる。オブジェクトの動作を以下に示す。

[オブジェクトの動作] [図2.2]

(1) オブジェクトは、メッセージを受信したとき、活性化される。

(2) メッセージは、オブジェクトの持つメソッドを示すセレクタを含む。

(3) 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、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。

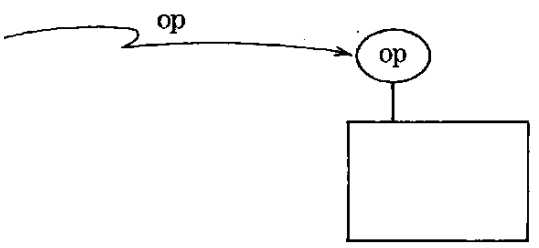


図2.2　メッセージパッシング

以上の機構をメッセージパッシング(message passing)という。メッセージパッシングは、遠隔プロシージャ呼び出し(RPC : Remote Procedure Call)と同じ機構である。例えば、データベースシステム [DATE,TAKI96] を考える。データベースシステムは、サーバとクライアントから構成される。サーバをオブジェクトと考えることができる。データベースサーバは、データベースに対する基本的な操作を行うものである。即ち、SQL をデータベースに対して、実行するとともに、データベースのインテグリティ制約を保ちながら複数の利用者の SQL を同時実行し、障害から復旧させる。これに対してクライアントは、ユーザとのインタフェースの機能を持つ。ユーザの応用プログラムは、クライアントで実行され、この中のデータベースに対する操作演算 (SQL) は、サーバに送信される。サーバは、データベースの操作演算を受信すると、これを実行し結果をクライアントに返す。以上のように、クライアントの応用プログラムから見ると、サーバを関数呼び出しすることにより、データベースを利用しているように見える。これを、遠隔プロシージャ呼び出し(RPC : Remote Procedure Call)という[図2.3]。サーバの SQL がメソッドである。

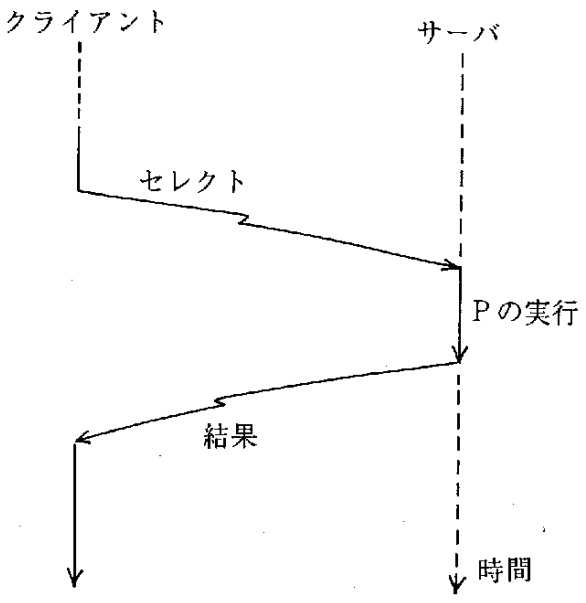


図2.3　遠隔プロシージャ呼び出し

2.1.3　階層化

複雑なシステムを構築したり、理解するためには、システムを構成するオブジェクトを

階層化することが必要である。階層化の方法として、まず抽象化がある。抽象化とは、下位層がより具体的なものを示し、上位がより抽象的をことを示すことである。システム内のオブジェクトは、抽象化の概念により階層化される。

A．クラスと例

同一の性質を持つオブジェクトの集合をクラスとする。クラスに属するオブジェクトをクラスのインスタンスとする。クラスとそのクラスに属するインスタンスオブジェクトの関係を instance_of という。図2.4に学生のクラスとそのインスタンスの関係を示す。クラス学生は、学生の持つ属性からなる。これのインスタンスは、各属性に値を与えたものである。

データベースシステムで考えると、クラスはテーブルのスキーマ(テーブル名、属性の構成)を示す。

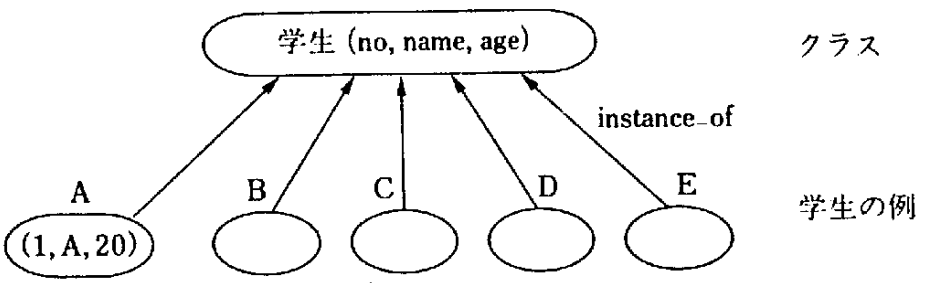


図2.4　クラスとインスタンスの関係

B．汎化

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が汎化 (generalization) である。抽象化された上位の概念と、より具体化された下位の概念の間の関係である。これを is-a の関係という。オブジェクトがクラスであるときに、上位のオブジェクトを上位クラス (super-class)、下位のオブジェクトを副クラス (subclass) ともいう。図2.5に、上位クラスのコンピュータと下位クラスのワークステーションの関係を示す。例えば、ワークステーションはコンピュータ(ワークステーション is_a コンピュータ)である。ワークステーションは、コンピュータとしての性質を持つこととなる。

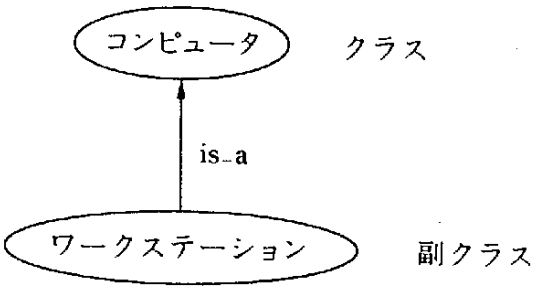


図2.5　階層化

C．集積化

集積化 (aggregation) は、 part_of といわれる関係である。あるオブジェクトが、他のオブジェクトからどのように構成されているかを示している。図2.6にコンピュータが、CPU、メモリ、ディスク、IO、バスから構成される part_of の関係を示す。

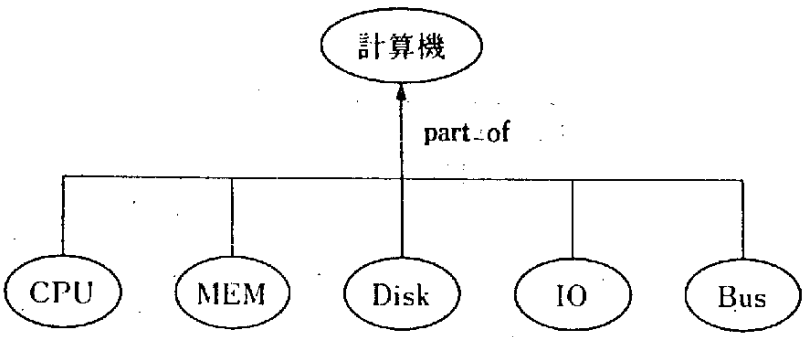


図2.6　集積関係

以上みてきたオブジェクトの関係 (クラスとインスタンス、 is_a、 part_of) は、互いに直行した関係にある。図2.7に、コンピュータについての例を示す。

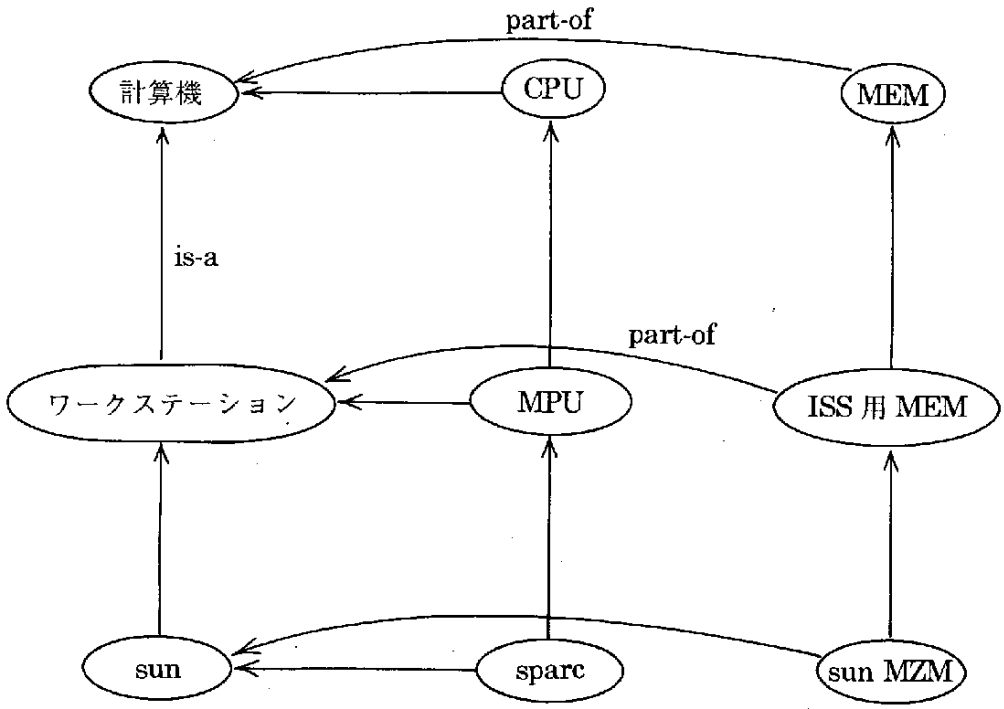


図2.7　is-a と part-of の関係

2.1.4　継承

オブジェクト A と B の間に is-a の関係があり、A が上位とする (B　is_a　A)。B は A である (B　is-a　A) ことから、上位のオブジェクト A の持つ性質、即ち、データ構造とメソッドを、下位の B も持つことになる。これを、A から B への継承 (inheritance) という。例えば、教育システムのソフトウェア開発を考える。A は教育システムの共通のソフトウェアモジュールとする。このとき、ある大学用の教育システムを B とする。B は、共通のモジュールを用いて、この大学に固有のソフトウェアを開発している。このとき、A 内のモジュールが B に継承することになる。

あるクラスが複数の上位のクラス $C_1,...,C_n$ を持ち、各上位クラス C_i から性質 p_i を継承する場合を多重継承とする。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。例えば、図2.8で、勤労学生 StudentWorker は、Emp と Student の両方の性質が継承される。ここで、Emp の bonus は給料に関していて、Student の bonus は成績に関している。これは、継承する性質が競合する例である。競合の解消方法としては、上位のクラスに優先度を与えて、優先度の高いクラスの性質が継承されるものとする

方法がとられている。この競合を、例外を示すために利用することも行われている。クラス間の汎化の関係が木構造をしているときは、多重継承は生じない。何故ならば、各クラスCは複数の上位クラスを持たないからである。これに対して、束である場合には、各クラスは一般に一つ以上の上位クラスを持てるので、多重継承の問題が生じる。

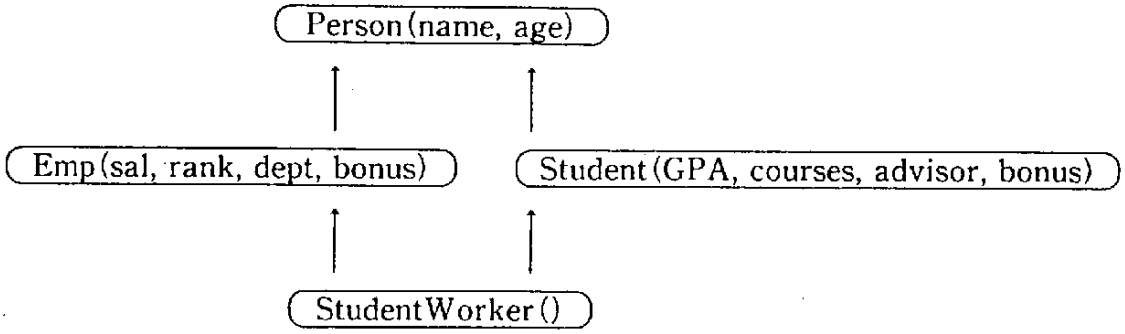


図2.8　多重継承

2.1.5　ポリモルフィズム

例として、数を示すオブジェクトNと論文を示すオブジェクトPを考える。おのおの印刷を行うメソッドを提供しているとする。このとき、同一名のメソッドprintを用いて、印刷を行えることが、ポリモルフィズム (polymorphism) である。オブジェクトのデータ構造の型を意識せずに、操作を行えることである。

2.2　分散システム

オブジェクト指向システムによる分散システムでは、プロセス、ファイル、ハードウェア、プログラム等のすべての資源をオブジェクトと考える。オブジェクトは、データとこれに対する操作演算を一まとまりとしたもので、利用者はオブジェクトが提供する演算を用いてのみオブジェクトを操作できる。オブジェクトは、データと演算を一体化したものであった。オブジェクトは、メッセージを受け付け、メッセージが示す演算を実行し、結果を返す。これは、遠隔プロシージャ呼び出し (RPC) である。このように、オブジェクトは、メッセージを受信して起動される。このことから、受動的 (passive) オブジェクト

とも呼ばれる。

これに対して、オブジェクトに、データ、演算、さらに演算を実行するプロセッサも含めるものを能動的(active)オブジェクト[LISK88, SCHA86, EPPI91]という。オブジェクト内のプロセッサは、演算の実行とともに、オブジェクト内部の資源の管理を行う。各オブジェクトがプロセッサを持つことから、オブジェクトを並行して実行できる。

分散システムの構成について考える。集中システムでは、一つのコンピュータ内にすべての機能が実現され、利用者は端末から利用するものであった。ワークステーション等の小型のコンピュータが通信ネットワークにより相互接続された分散システムでは、システムの機能を、サーバとクライアントに分割し、各々を異なったコンピュータに配置する機能分散型の形態が考えられてきている[図2.9]。これを、クライアントサーバモデルという。

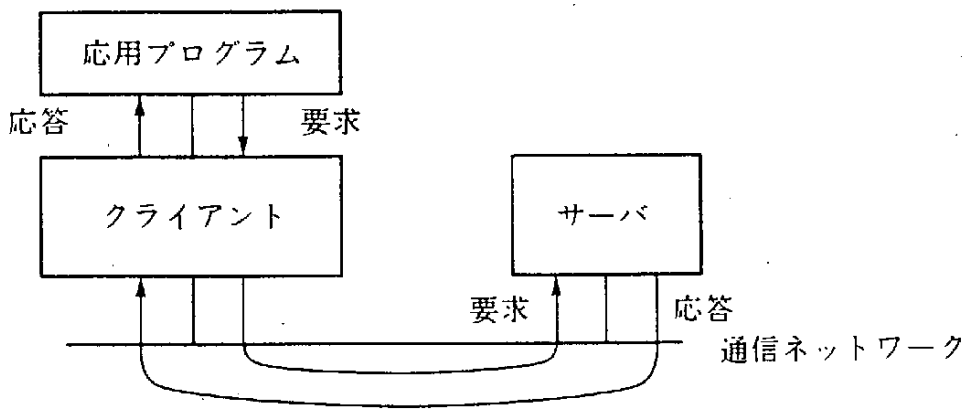


図2.9　クライアントサーバモデル

A．クランイアントサーバモデル

クライアントサーバモデルについて考察する。クライアントとサーバは以下にまとめる。

(1) サーバとは、あるサービスを提供するシステムであり、サービスとしては、データベース、ファイル、プリンタ、通信ゲートウェイといったものがある。例えば、データベースサーバは、データベースに対する検索と更新の操作演算を提供する。

(2) クライアントは、サーバと利用者間のインタフェースであり、サービスに依存している。例えば、データベースシステムでのクライアントでは、利用者のプログラムが実行される。このプログラムがデータベース操作演算を実行すると、クライアントは、サーバにこの操作演算を発行し、サーバで演算が実行される。

サーバは、演算を実行するオブジェクトと考えられる。

B．遠隔プロシージャ呼び出し (RPC)

サーバの提供するサービスは、演算の集合として与えられる。クライアントは、サーバに演算の実行を依頼する。サーバは、この演算を受け付けられるならば、演算を実行し、結果をクライアントに返す。このように、サーバは、演算をメソッドとしたときのオブジェクト指向システムでのオブジェクトと考えることもできる。このようなサーバとクライアント間の通信を遠隔プロシージャ呼び出し (RPC) という。

複数のコンピュータ内のオブジェクト間で通信を行うには、オブジェクトは、メッセージの通信を送信 (send) と受信 (receive) といった通信命令を用いて行う必要がある。分散システムの目的の一つは、複数のコンピュータから構成されていながら、利用者にはあたかも一つのシステムのように見せることである。このためには、コンピュータ間での send と receive といった通信命令を、利用者プログラムに利用させないことが望ましい。図2.10では、サーバのオブジェクトPを、クライアント内のプログラムAが利用する例を示している。プログラムAは、通常の関数呼び出しと同じ呼び出し形式で、遠隔のサーバ内のプログラムPを実行できる。この呼び出しは、実際にはクライアントが send と receive といった通信命令を用いてサーバと通信を行い実現される。遠隔プロシージャ呼び出し (RPC) は、同期型通信であるので、利用者プログラムがサーバに演算を送信した後、応答が返ってくるまで待つ。RPCは、分散システムのソフトウェアを実現する場合の基本となる計算機構である。

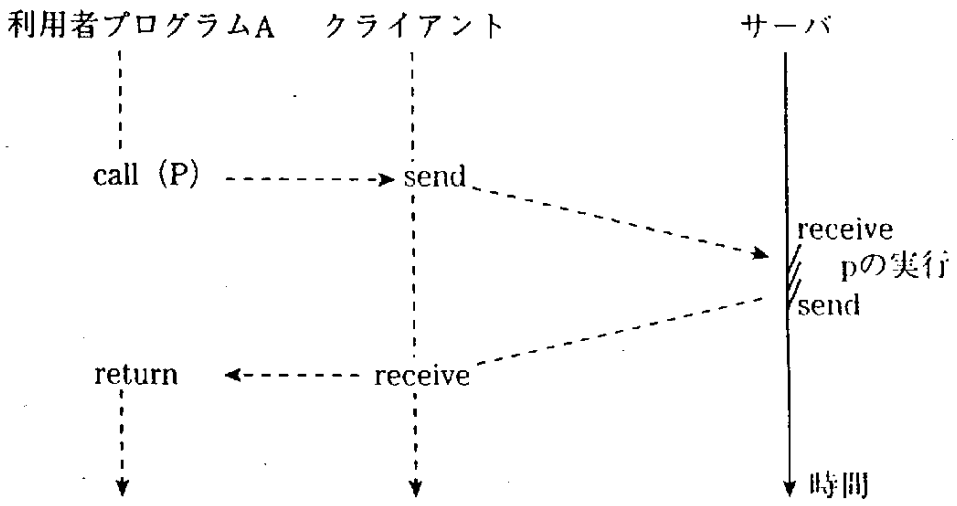


図2.10　遠隔プロシージャ呼び出し

C．遠隔プロシージャ呼び出しの信頼性

遠隔プロシージャ呼び出し(RPC)では、要求がクライアントからサーバに送信され、サーバからその応答が送信される。通信ネットワークとクライアントまたはサーバの故障から、要求と応答メッセージが宛先に届かない場合がある。例えば、クライアントCがサーバSに要求opを送信したのに、CがSからopの応答を受信しなかったとする。このため、Cが再度opをSに送信したとする。実際に最初の要求opメッセージがSに届かなかった場合には、Sがopを実行した上で応答をCに送信する。しかし、Sが最初のopを受信し、実行し、応答をCに返したが、応答メッセージがCに届かなかった場合が考えられる。ここで、Sが再度opを受信し、実行することになる。opが例えばデータベースシステムの検索演算であればopを何回実行しても、データベースに影響がない。しかし、opが更新演算である場合には、opが複数回実行されてしまい、データベースの状態が正しくなくなってしまう。このために、opの遠隔プロシージャ呼び出しの信頼性について、以下のセマンティクスが考えられる。

(1) 不定(maybe)：opが何回も実行されるかもしれない。
(2) 一回以上(at least one)：opが少なくとも一回は実行されることを保証する。サーバSが故障した場合には、opの実行は保証されない。
(3) 一回以下(at most one)：opは、高々一回実行される。サーバが故障しない場合には、opは一回だけ実行される。サーバが故障した場合でも、opは高々一回実行される。
(4) ただ一回(exactly once)：opは、ただ一回だけ実行される。

以上の遠隔プロシージャ呼び出しの信頼性を達成するためには、図2.11(a)に示すように、要求と応答のおのおののメッセージに対して、その受信確認(Ack)を送り返す方法がある。この方法を用いると、メッセージ数が増加するので、図2.11(b)に示すように、サーバSからの応答メッセージに要求メッセージのAck情報を含ませる方法もある。

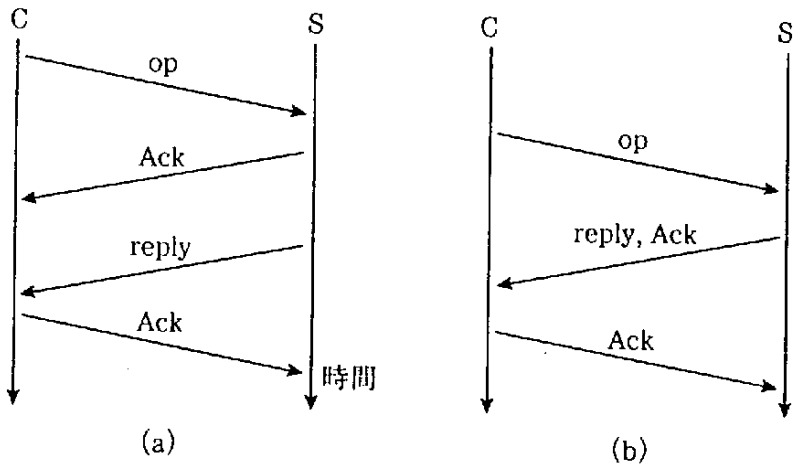


図2.11　遠隔プロシージャ呼び出しの信頼性

サーバは演算 op の処理要求を受信し、実行した上で応答をクライアントに返す。このとき、 op の処理の状態を保持しているかどうかにより、サーバには以下の種類がある。

(1) 状態なし (stateless) サーバ。

(2) 状態付き (statefull) サーバ。

状態なしサーバでは、 op の応答を返すと op についての全情報がなくなる。応答がクライアントに届かない場合には、クライアントが op の要求を再送してくる場合がある。このときには、サーバは op を新たに実行し、応答をクライアントに返す。

状態付きサーバでは、応答がクライアントに届いたことが確認されるまで、 op の処理を終わらさない。応答がクライアントに届かず、クライアントが要求を再送してきた場合には、 op を再実行せずに応答をクライアントに再送する。

2.3　CORBA

次に、オブジェクト指向概念に基づいた CORBA (Common Object Request Broker Architecture) [MOWB95] について説明する。現在の情報システムは、複数のコンピュータが通信ネットワークにより相互接続された形態となっている。通信ネットワークとしては、 TCP/IP が業界標準として、 Unix のワークステーション、 Windows, NT のパソコン (PC) を含めて広く利用されている。こうした共通のオペレーティングシステム (OS) と通信ネットワークとしての基盤技術の上に、リレーショナルデータベースシステム、

ワープロ、Notes等の種々の応用が実現されてきている。いわゆるミドルウェア(middleware)である。こうしたシステムは、クライアントサーバモデルにもとづき、クライアントとサーバから構成されており、クライアント側から、種々のサーバを容易に利用できなければならない。すなわち、種々のミドルウェアを用いて、利用者の必要とするアプリケーションを容易に実現し、運用できることが重要である。

CORBAは、OMG (Object Management Group)により開発された。ミドルウェアに対する共通の枠組みを与えるものである。OMGは、企業のコンソーシアムであり、ソフトウェアの相互運用可能なインタフェースを定義することを目的としている。CORBAでは、分散システムをオブジェクト指向概念を用いてモデル化している。CORBAは、インタフェースを定義するための言語として、OMG IDL (Interface Definition Langu-age)を提供している。

2.3.1 オブジェクト管理アーキテクチャ

OMGのオブジェクト管理アーキテクチャ[OMG93]について考察する。オブジェクト管理アーキテクチャ［図2.12］では、オブジェクト要求プローカ(ORB：Object Request Broker)中心となる。ORBは、システムの通信インフラとして、オブジェクト要求(request)を通信する機能を持つ。このとき、異なった計算環境、例えば、OSが異なっている、ワープロが異なっているといった異種の計算環境間でも、オブジェクト要求を通信する。

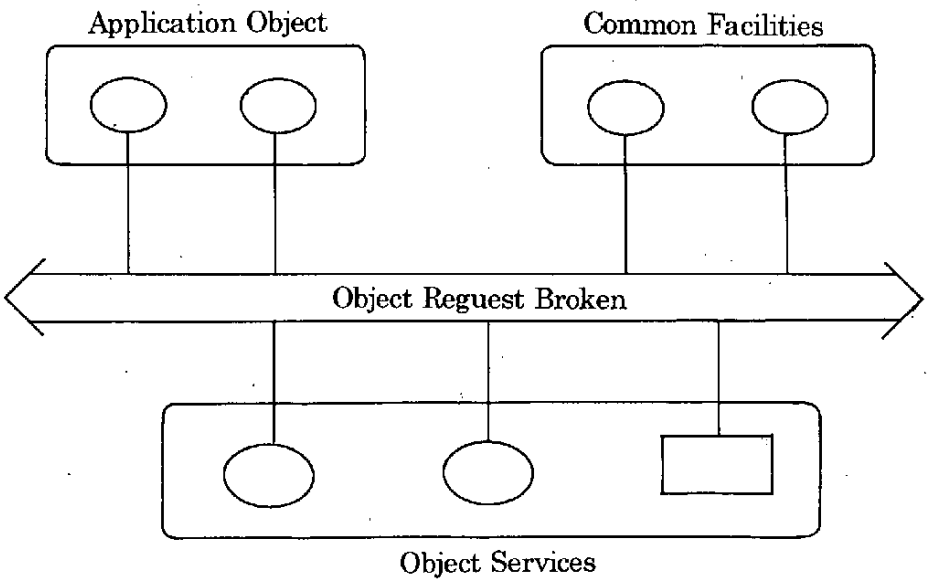


図2.12 オブジェクト管理アーキテクチャ

2.3.2　オブジェクト要求ブローカ(ORB)

CORBA は、アプリケーションレベルでの通信インフラの標準仕様を与えている。アプリケーションは、以下の二つの機構により、互いに通信を行うことができる。

(1) 静的インタフェース、

(2) DII (Dynamic Invocation Interface)。

インタフェースレポジトリ (Interface Repository) には、OMG IDL インタフェースが記憶されオンラインでアクセスできる。基本オブジェクトアダプタ (BOA : Basic Object Adapter) は、オブジェクトの実装のための ORB インタフェースの初期集合が記憶される。

CORBA では、すべてのアプリケーションをオブジェクトととらえている。CORBA は、これらのオブジェクト間の一対一(peer-to-peer)の通信をサポートしている。オブジェクトは、クライアントでもあり、サーバでもある。これらの関係は、応用間により定まるものである。オブジェクトが他のオブジェクトから呼び出される (invocation) ときには、このオブジェクトはサーバとなる。一方、呼び出しているオブジェクトは、クライアントとなる。サーバのオブジェクトは、オブジェクト実装 (object implementation) と呼ばれる。

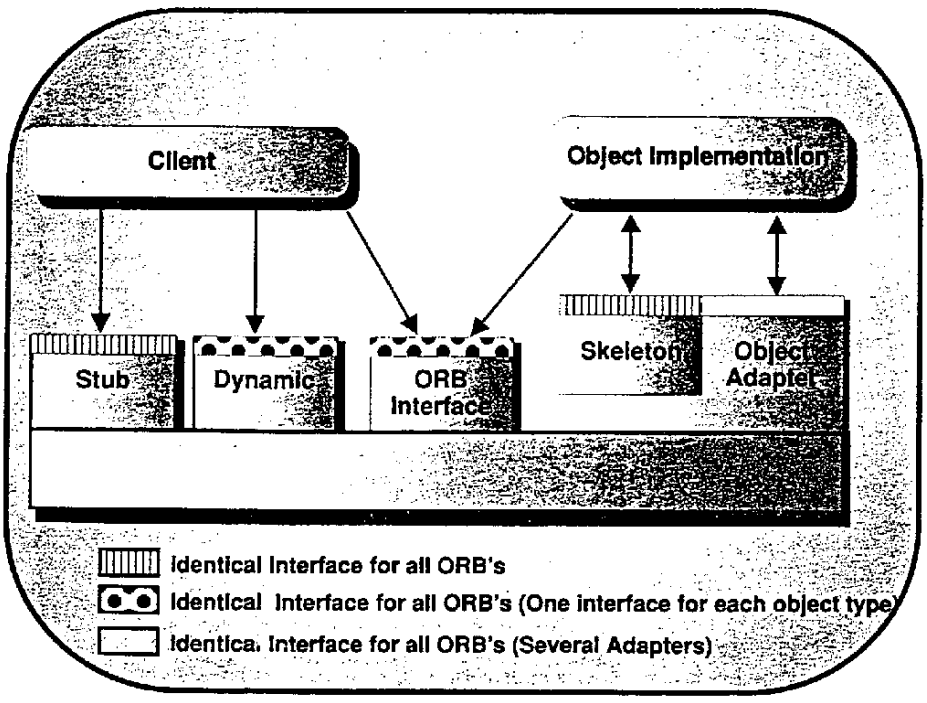


図2.13　CORBA インタフェース

2.3.3 IDL

OMG IDLは、オブジェクトのインタフェースを定義するための言語である。OMG IDLを用いて、属性と演算シグニチャから構成されるインタフェースが定義される。また、オブジェクトの再利用を行うために、インタフェース記述間での継承機能もサポートしている。

2.3.4 オブジェクトサービス

オブジェクトサービスは、システムサービスインタフェースの集合である。OMGオブジェクトサービス(Object Service)は、共通オブジェクトサービス仕様(COSS: Common Object Service Specification)から構成される。COSSは、複数のボリューム(volume)から構成される。各ボリュームは、一つのRFP (Object Services Request for Proposal)に対応している。現在、RFP1, EFP2, RFP3, RFP4, RFP5のRFPが提供されている。

RFP1

(1) Object Event Notification Service

(2) Object Life Cycle Service

(3) Object Naming Service

(4) Object Persistence Service

RFP2

(1) Object Concurrency Service

(2) Object Externalization Service

(3) Object Relationships Service

(4) Object Transaction Service

RFP3

(1) Object Security Service

(2) Object Time Service

RFP4

(1) Object Licensing Service

(2) Object Properties Service

(3) Object Query Service

RFP5

(1) Object Change Management Service

(2) Object Collection Service

(3) Object Trader Service

(4) Object Startup Service

3章　インターネット上のオブジェクト

インターネットの普及により、世界中のコンピュータが相互接続されてきている。一方、オペレーティングシステム(OS)も、Windows, NT, Unixといった共通のプラットフォームが整い、コンピュータ相互の運用性が高まってきている。こうした情報技術基盤の整備にともない、ミドルウェアの標準化が大きな課題となっている。ここでの標準化は、これまでのISO, JISといった公的機関によるものではなく、TCP/IP, WWWのように業界標準が主導していくものと思われている。情報システムも、これまで仕様書を作成し、コーディングを行うことで、開発、保守、運用を遂行していたが、こうしたシステムの開発方法も変化しつつある。たとえば、インターネット上の無数のソフトウェア資源を利用して、自分の必要とするシステムを組み上げていく方法である。このため、本調査研究では、インターネット上のソフトウェア資源の調査を行ったが、膨大な数の資源があることから、ここでは、以下に特定し、その評価を行った。

(1) e-mail

(2) Notes

(3) データベースシステム

これらを、表3.1にまとめる。

表3.1　インターネット上のソフトウェア資源の評価

オブジェクト作成	エディタ メールとWebの結合	Netscape navigator
	オブジェクトとツールの関連付け フォーマット変換(テキスト、Jpeg, html等) 圧縮、コード変換	Netscape navigator AL-Mail, Eudora, Netscape navigator
	電子署名 発信者ログイン認証	PGP PGP, PEM
オブジェクト管理	キーワード設定 オブジェクト間のリンク設定 発信・不達メールリスト フォルダ管理(作成、削除、内容追加/削除、構造化リンク)	 AL-Mail,Eudora,Netscape

宛先リスト作成	宛先リストの作成 宛先のグループ化	AL-Mail,Eudora,Netscape
宛先指定	1対1 1対多 速達、進展、回覧、配布先添付 発信宛先エラー表示 エラー宛先修正 受信者許可リスト	全mailer Notes
発信	発信、再発信 発信取消 発信日指定 発信メールリスト 発信状況表示、ログ 返信(往復メール複写) 転送 FAXでの送信	全mailer 全mailer 全mailer 全mailer 全mailer
受信	メール受信 受信確認の受信 着信通知(音等)	全mailer 全mailer
自動分類	subject, キーワード等での分類	全mailer
自動転送	留守番転送	
内容表示	ビューア(テキスト、図、音声) オブジェクトとツールの関連付け 受信メールリスト 表示オプション(到着順等) FAXからの着信	 全mailer
オブジェクト管理	キーワード認識 オブジェクト間の認識 修正、参照	
掲示板		
登録	オブジェクト作成 (テキスト、図等) 登録、削除 緊急表示、緊急通知 キーワードの自動抽出 構造化、ハイパーリンク	Internet assistant (htmlオーサリングツール)
参照、検索	参照、検索 フィルタリング	Webブラウザ
管理	掲示板作成 アクセス権設定 参加者の認証	Netscape server, Information server

4章　信頼性、可用性技術

分散システムは、複数のオブジェクトから構成される。分散システム上の応用(分散アプリケーション)は、これらの複数のオブジェクトの協調動作により実現される。これらのオブジェクト間の信頼性と可用性を向上させるための技術について考える。

4.1　分散システム

分散システムでは、複数のコンピュータ内のオブジェクト間での協調動作により処理が行われる。複数のオブジェクト間で行われる計算を分散計算(distributed　computation)という。

4.1.1　分散システムの状態

A．オブジェクトのモデル

集中システムでは、プログラムは一つのコンピュータ内で処理される。これに対して分散システムでは、利用者のプログラムは複数のコンピュータにより分散して処理される。ここで、各コンピュータでのオブジェクトの実行状態について考える。オブジェクトは、変数の値の集合で与えられる状態を持ち、メッセージの受信、命令実行等の事象の生起により状態が遷移していく。このことから、オブジェクトを有限状態機械として示せる。事象としては、内部的な計算等を行う内部事象と、通信事象がある。通信事象には、メッセージの送信と受信を示す事象がある。オブジェクトは、各コンピュータで逐次処理されることから、オブジェクトは事象の系列としてモデル化する。ここで、オブジェクトp_iの初期状態をs_i^0とする。事象e_i^1により状態s_i^1に遷移する。このように、状態s_i^jは、事象e_i^{j+1}により状態s_i^{j+1}に遷移する[図4.1]。e_i^{j+1}をp_iのj+1番目の事象とする。オブジェクトp_iの事象の系列$<e_i^1, e_i^2, ...>$を、p_iの履歴h_iとする。初期状態からj番目の事象e_i^jの系列$<e_i^1, ... e_i^j>$を、h_i^jと書くことにする。

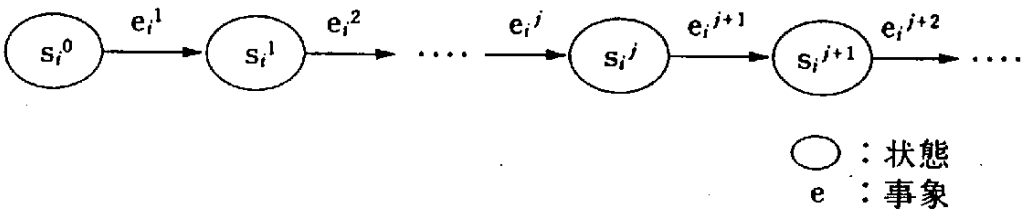


図4.1　有限状態機械

B．事象間の因果順序

プログラムが複数のコンピュータで実行されることから、異なったオブジェクトの事象の間でどちらが先に起きたかを決めることが重要となる。集中処理システムでは、各オブジェクトはコンピュータが持つ時計を参照することにより、コンピュータ内での時刻を知ることができる。事象の起きた時刻により、事象を順序つけることができる。これに対して、分散システムでは、各コンピュータが時計を持つがこれらの時計が同一の時刻を示すことを保証できない。このため分散システムでは、各オブジェクト内での事象がいつ起きたかということよりも、事象がどのような順序で起きたかという事象間の因果関係(causality)が重要である。ここで、事象間の因果関係を示す半順序関係→を以下のように定義する [LAMP78]。

[因果関係] e_1 と e_2 を任意の事象とする。以下のいずれかが成り立つとき、e_1 は e_2 に因果先行する ($e_1 \rightarrow e_2$)。

(1) e_1 と e_2 が同じオブジェクト内で起き、e_1 が e_2 が先行する。

(2) e_1 があるメッセージ m の送信を示す事象であり、e_2 が m の受信を示す事象である。

(3) $e_1 \rightarrow e_3$ で $e_3 \rightarrow e_2$ なる事象 e_3 が存在する。

e_1 と e_2 間に、$e_1 \rightarrow e_2$ と $e_2 \rightarrow e_1$ のいずれも成り立たないとき、e_1 と e_2 は、同時に起きるとし、$e_1 || e_2$ と書く。ここで、任意の事象 e_1 に対して、$e_2 \rightarrow e_1$ なる事象 e_2 の集合を e_1 の因果錘 (causality cone) とし、cone(e1) と書く。

[例] 図4.2に、3つのオブジェクト p_1、p_2、p_3 内の事象の因果関係の例を示す。ここで、記号 a、b、...、r、s、...、y、z は事象を示す。p_1 内で、a は b よりも先に起きるので、a → b である。p_1 の履歴 h_1 は <a, b, c, d> である。また、h_1^1 = <a>、h_1^3 = <a, b, c> である。b は p_1 のメッセージ m の送信事象を示す。また、s は p_2 での、x

は p_3 での m の受信事象を示す。ここで、b → s、b → x である。同様に、u → y である。このことから、a → b → u → y がいえる。即ち、事象 y は、a よりも後に起きたことになる。一方、事象 c と s 間には→の関係がないので、同時に起きたといえる (c| |s)。

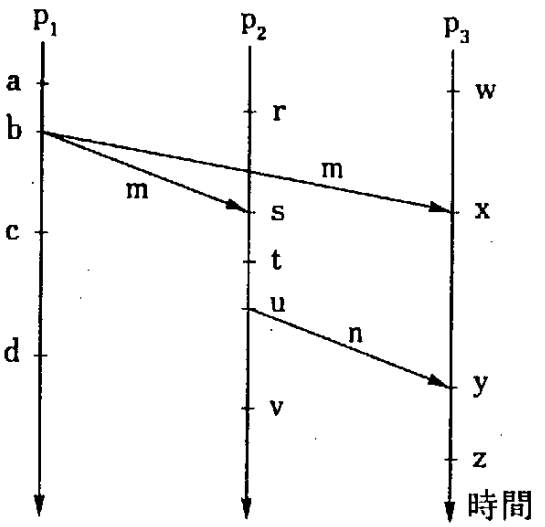


図4.2　因果関係

C．状態

分散計算が n 個のオブジェクト $p_1,...,p_n$ により行われているとする。図4.2に示すように、分散計算は、各オブジェクト内の事象の集合と、これらの事象間の因果順序により示せる。即ち、E を $p_1,...,p_n$ 内の事象の集合としたとき、分散計算は半順序集合 $<E, \rightarrow>$ である。

分散システムでは、各オブジェクトがどのような状態にあるかを知ることが重要となる。集中処理システムでは、全オブジェクトが一つのコンピュータ内にあるので、ある時刻の各オブジェクトの状態を知ることができる。しかし、分散システムでは、各コンピュータの時計が同一の時刻を示せないことから、ある時刻の状態を集めることができない。従って、分散計算ではある時刻の状態ではなく、各オブジェクト間で矛盾のない状態の集合を分散計算の状態として考える必要がある。ここで、分散計算 D の状態を各オブジェクト p_i の状態 $s_i^{j_i}$ の組 $<s_1^{j_1},...,s_n^{j_n}>$ とする。D の初期状態は、$S_0 = <s_1^0, ..., s_i^0, ..., s_n^0>$ である。あるオブジェクト p_i の事象 e_i^1 により、D は状態 S_0 から状態 $S_1 = <s_1^0, ..., s_i^1, ..., s_n^0>$ に遷移する。このように、分散計算の状態は、各オブジェクトの事象により遷移する。こうして得られる分散計算の状態が無矛盾であるとは何かについて考える。

D．カット

ここで、履歴h_1^{j1}, ..., h_n^{jn}の集合をカット(cut) Cとする。カットCは、分散計算状態$<s_1^{j1},...,s_n^{jn}>$を示している。各p_iの履歴$h_i^{ji}=<e_i^1, ..., e_i^{ji}>$の最後の事象$e_i^{ji}$の組$<e_1^{j1}, ..., e_n^{jn}>$をCの前線(frontier)とする。図4.2で、{<a, b, c>, <r, s, t, u>, <w, x, y, z>}はカットC_1であり、<c, u, z>が前線である。図4.3にカットC_1を実線で示す。また、図4.3に、<c, t, z>を前線とするカットC_2を点線で示す。カットC_1では、メッセージnの送信事象uと受信事象yが前線の事象に因果先行している。しかし、C_2ではnの送信事象uよりも前線tが因果先行している。即ち、C_1を考えるとnの送信と受信の両方の事象が起きていることがわかるが、C_2ではnの送信事象なしに受信事象が起きていることとなる。このことから、C_2は矛盾しているといえる。

カットCは、次の条件を満足するときに、無矛盾である。C内の任意の事象e_1とe_2に対して、e_1がCに含まれ、$e_2 \rightarrow e_1$ならば、e_2もCに含まれる。

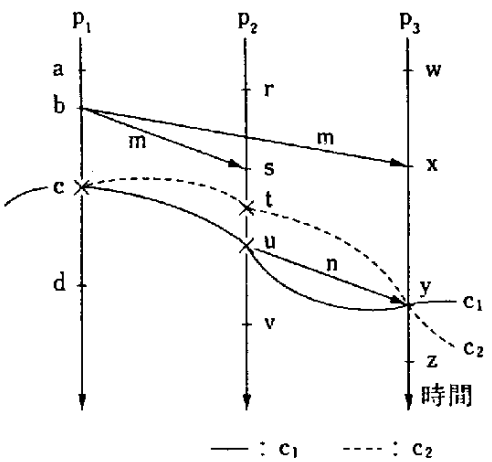


図4.3　カット

分散システムでは、無矛盾なカットを見つけることが必要となる。

4.1.2　事象の順序付け

分散システムでは、事象の因果順序を決定することが重要である。この問題について考える。

A．線型時間

まず、Lamport[LAMP78]により示された論理時間について述べる。各オブジェクトpiは、以下のようなローカル時刻を示す変数hiを持つ。

(1) 各p_iは、事象が起きる毎に、h_iを増加させる。

(2) p_iがメッセージmを送信する時、h_iのローカル時刻をmに与える。ここで、time(m)は、mに与えられたローカル時刻を示すとする。

各オブジェクトp_iのローカル時刻h_iは、以下の規則により変更される。

[規則L_1] p_iで各事象(内部事象、送信と受信事象)が起きたときに、$h_i = h_i + d$ $(d > 0)$とする。

[規則L_2] 各p_iが、p_jからメッセージmを受信したとする。このとき、$h_i = \max(hi, time(m)) + d$とする。

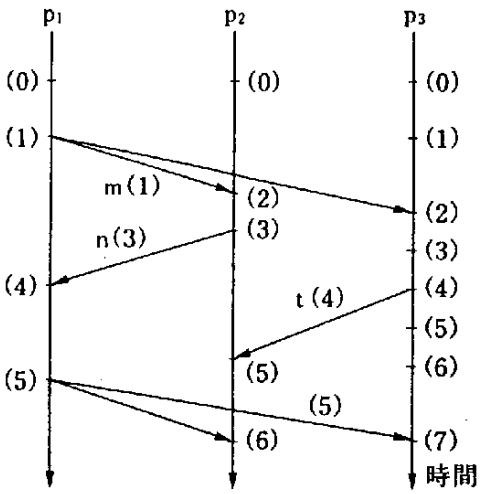
オブジェクトp_i内で、事象eが起きたときのローカル時刻をh_iとする。このとき、eに対して値の対$<i, h_i>$をeの時刻印(time stamp) ts(e)とする。分散システム内の事象を、時刻印により順序付ける。ここで、事象e_1とe_2の時刻印を各々$ts(e_1) = <i_1, t_1>$と$ts(e_2) = <i_2, t_2>$とする。このとき、以下のように、e_1とe_2を順序つけ、e_1はe_2より先行する$(e_1 \Rightarrow e_2)$とする。

[事象の順序]

(1) $t_1 \leqq t_2$であるか、または (2) $t_1 = t_2$で$i_1 < i_2$であるときに、$e_1 \rightarrow e_2$とする。

ここで、$e_1 \Rightarrow e_2$ならばe_1はe_2に因果先行する$(e_1 \rightarrow e_2)$。$e_1 \| e_2$であっても、$e_1 \Rightarrow e_2$または$e_2 \Rightarrow e_1$のいずれかである。以上のLamportの論理時間を用いると、分散システム内の全事象を全順序づけられる。このため線型時間と呼ばれている。

[例] 図4.4に、3つのオブジェクトp_1、p_2、p_3についての例を示す。ここで、$d = 1$である。各オブジェクトのローカル時刻の初期値は0であるとする。p_1は、メッセージmをp_1とp_2に送信するときに、送信事象の時刻印$h_1(=1)$を含ませる。p_2がmを受信すると$h_2 = 0$なので、$h_2 = 1$として、1を加えて$h_2 = 2$とする。p_3も同様に、mを受信すると$h_3 = 2$となる。以下同様にして、各オブジェクトの論理時間が増加していく。p_3がメッセージn $(h_3 = 4)$をp_2に送信する。p_2で、mの時刻印は2でnは4であるので、mがnに因果先行することがわかる。



(i)：ローカル時刻 i

図4.4　線型時間

B．ベクトル時間

次に、ローカル時刻を半順序つけるベクトル時間 [MATT89] について述べる。各オブジェクト p_i の各ローカル時刻は、n 次元のベクトル $v_i = <v_i[1],...,v_i[n]>$ により示される。各 $v_i[j]$ は、p_i からみた p_j のローカル時刻である。特に $v_i[i]$ は p_i 自身のローカル時刻である。v_i は、p_i 内の各事象 e に与える時刻印 ts(e) となる。v_i は、以下の規則により変更される。

[規則 V1] 各 p_i で、事象 e が起きる前に $v_i[i] = v_i[i] + d$ $(d > 0)$ とする。

[規則 V2] p_i が、p_j からメッセージ m を受信したとする。m は、p_j が m を送信したときのローカル時刻を示すベクトル v_j を含む。このとき、各 h $(=1,...,n)$ に対して、$v_i[h] = \max(v_i[h], v_j[h]) + d$ とする。

[例] 図4.5に、3 つのオブジェクト p_1、p_2、p_3 についてのベクトル時間の例を示す。この例も、$d = 1$ である。各オブジェクトのローカル時刻は、はじめに、<0, 0, 0> であるとする。p_1 がメッセージ m を送信するとき、ローカル時刻 $v_1 = <1, 0, 0>$ を m に含めて p_2 に送信する。p_2 は m を受信したとき、$v_2 = <1, 1, 0>$ となる。以下同様に、各オブジェクトのローカル時刻が変化していく。

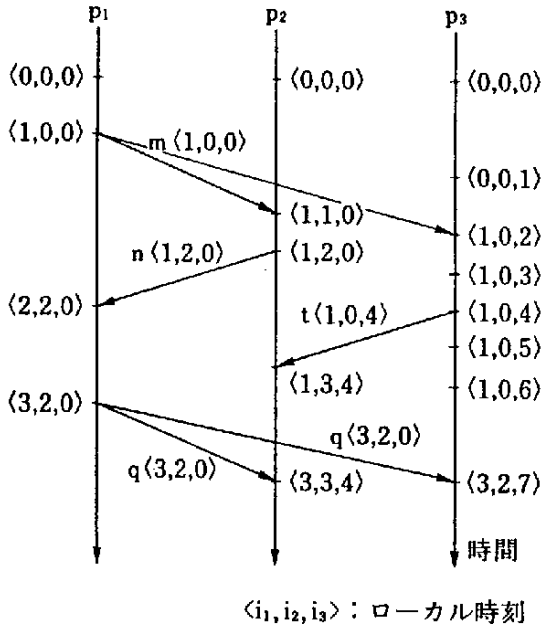


図4.5　ベクトル時間

ローカル時刻間の順序について考える。v_i と v_j を二つのローカル時刻とする。このとき、v_i と v_j 間に以下の順序を定義する。

[ベクトル時刻間の順序]

(1) 各 h (= 1,...,n) について $v_i[h] \leqq v_j[h]$ であるならば、$v_i \leqq v_j$ である。

(2) $v_i \leqq v_j$ でかつ、 ある h について、$v_i[h] \neq v_j[h]$ であるならば、$v_i < v_j$ である。

(3) $v_i < v_j$ でも、$v_i > v_j$ でもないならば、$v_i \| v_j$ である。

例えば、図4.1で、<1,2,0> ≦ <3,2,0> である。以上のローカル時刻間の順序関係<を用いて、分散システム内の事象 e_1 と e_2 間の先行順序 (⇒) を以下のように定義する。ここで、e_1 と e_2 は、各々オブジェクト p_i と p_j で起き、時刻印 $ts(e_1) = <i, v_i>$ と $ts(e_2) = <j, v_j>$ を持つとする。

[事象間の順序]

(1) $v_i[i] < v_j[i]$ であるならば、$e_1 \Rightarrow e_2$ である。

(2) $v_i[i] < v_j[i]$ でかつ、$v_i[j] < v_j[j]$ あるならば、$e_1 \| e_2$ である。

ここで、任意の二つのメッセージ m_1 と m_2 に対して $m_1 \rightarrow m_2$ ならば、かつこのときに限り、$m_1 \Rightarrow m_2$ である。図4.1で、メッセージ m の時刻印は <1, <1,0,0>> であり、n は

<2, <1,2,0>> である。よって、 m ⇒ n である。一方、 p の時刻印は <3, <0,0,4>> であるので、 m | | p である。ベクトル時間は、 ISIS[BIRM91] で用いられている。

事象 e の時刻印 $ts(e) = <v_i, i>$ で、 $v_i[i]$ は、 p_i で起きた事象のなかで、 e に先行する事象の数である。従って、$\Sigma_{j=1,\ldots,n} v_i[j]-1$ は、 e に先行する事象の総数である。

分散システムでは、オブジェクトから送信されたメッセージ m は、宛先のオブジェクトに必ず届くとは限らない。例えば、通信ネットワーク内の障害により、 m が紛失する場合がある。オブジェクト p_i が二つのメッセージ m_1 と m_2 を受信したときに、 $m_1 \rightarrow m_3 \rightarrow m_2$ なるメッセージが存在するかどうかを決めれるならば、メッセージの紛失を検出できる。この問題は、ギャップ検出 (gap detection) といわれる。これまで述べた線型時間とベクトル時間を用いた場合には、メッセージの紛失（ギャップ）を検出できない。このために、文献 [NAKA94] により送信オブジェクトがメッセージに通番を与え、これを用いることによりメッセージの紛失を検出できる方式が示されている。

C．並行度

分散システム S で、どの程度事象が同時に起きるかについて考える。ここで、 E を S 内の全事象の集合とする。ここで、以下を定義する。

$Con(S)=|\{ <e_1, e_2> | e_1, e_2 \in E$ で、 $e_1 | | e_2\}|/$

$|\{<e_1, e_2> | e_1, e_2 \in E$ で、 e_1 と e_2 は、異なったオブジェクトの事象$\}|$

二つのシステム S_1 と S_2 に対して、 $Con(S_1) > Con(S_2)$ ならば、 S_1 は、 S_2 よりも並行実行度が高い。

4.2 排他制御

分散システムでは、複数のオブジェクト $p_1,\ldots,p_n$ により、ある資源 x が利用されることがある。例えば、 p_1 が x を read し、 p_2 が write する場合では、 p_1 と p_2 が同時に x を利用できない。このとき、 p_1 と p_2 は x を競合して利用するという。このように複数のオブジェクトが x を競合して利用するとき、あるオブジェクト p_i に x を利用させて、他のオブジェクトに x を用させないことにより競合が解消される。このとき、 p_i は、 x に対す

る特権を持つ、またはxを獲得しているとする。特権を持たない他のオブジェクトはxを利用できない。オブジェクト内で、競合資源を操作する部分を危険領域(critical region)という。集中処理システムでは、セマファ[TANE92]等を用いて、特権を与える制御が行われる。分散システムでは、複数のコンピュータでオブジェクトが動作するために、オブジェクト間での通信が必要となる。

4.2.1 集中処理方式

まず、集中処理方式について考える。あるオブジェクト p_0 が指揮オブジェクトとして、どのオブジェクトに資源xについての特権を与えるか決定する。

[集中処理方式][図4.16]

(1) 資源xを利用しようとするオブジェクト p_i は、まず、p_0 にxの操作要求を出す。p_0 からの応答を持つ。

(2) p_0 は、高々一つのオブジェクトがxの特権を与える。xの特権をどのオブジェクトにも与えていなければ、p_0 は p_i に特権を与える。

(3) p_0 からxの特権を得たならば、p_i はxの操作を行う。xについての処理を終了したならば、p_0 に通知する。

以上の方式は、指揮オブジェクト p_0 が故障した場合に他のオブジェクトが待ち状態となる欠点がある。

図4.6 集中処理方式

4.2.2 分散計算方式

指揮オブジェクトが存在しない分散計算方式について考える。p_1, ..., p_n が資源xを利

用する場合を考える。ここでは、各オブジェクトのローカル時刻を用いて、競合資源に対する複数のオブジェクトからの操作要求を順序付け、この順序に従って、オブジェクトに利用させる方法について述べる [RICA81]。各オブジェクト p_i のローカル時刻を h_i とする。ローカル時刻としては、前項で述べた時刻を用いる。各オブジェクト p_i は、x を利用する前に、x をりようすることの要求メッセージ m を全オブジェクトに送信する。m は、p_i のローカル時刻を含み、time(m) は
このローカル時刻を示すとする。

次に、オブジェクト p_j が、p_i から、以上のメッセージ m を受信したとする。p_j は、以下の動作を行う [図4.7]。

[分散計算方式]

(1) p_j が x を利用していないならば、OK メッセージを p_i に送信する。

(2) p_j が x を利用しているならば、応答を返さない。m を待ち行列 Q_j に格納する。

(3) p_j が要求メッセージ m' を送信しているならば、time(m) と time(m') の比較を行う。time(m) < time(m') ならば、OK を p_i に送信する。time(m) > time(m') ならば、p_j は m を待ち行列 Q_j に記憶する。

p_i は、要求メッセージ m を送信した後、他の全オブジェクトからの応答を待つ。

(1) p_i が、全オブジェクトから OK を受信したならば、x を利用する。

(2) p_i が、x を利用し終わったならば、待ち行列 Q_i にあるメッセージを送信したオブジェクトに OK を送信し、Q_i から除去する。

以上の手順を用いると、2(n - 1) 個のメッセージが送信される。以上の手順も、あるオブジェクトの停止に対して残りのオブジェクト方式が待ちとなる問題がある。

図4.7には、オブジェクト A が資源 x を利用する場合を示している。まず A は、三つのオブジェクト B, C, D に、要求メッセージを送信する。B, C, D が x を利用していないならば、OK メッセージを送信する。A は、3 つのオブジェクトから OK を受信すれば、x を利用する。

図4.7　分散計算方式

4.2.3　トークンリング方式

他に、オブジェクトをリング状に結合する分散計算方式がある[図4.8]。リング内で、オブジェクトは $p_1,...,p_n$ という順序で結合され、p_n は p_1 と結合されている[図4.8]。リング内で、常にただ一つのオブジェクトにトークンtが与えられる。トークンは、リングに沿って、p_i から p_{i+1} にわたされ (i=1,...,n-1)、p_n から p_1 に次々に渡される。トークンを獲得しているオブジェクトが、特権を持つ。p_i がトークンtを受信したとする。

(1) p_i が資源xを利用しようとしているならば、xを利用する。xの利用が終了したときに、トークンtを p_{i+1} に渡す。

(2) p_i がxを利用しないならば、トークンtを p_{i+1} (i=nならば、p_1) に渡す。

本手順では、トークンtがリングを一周する間に各オブジェクトは一回はxを利用できる。トークンが紛失した場合は、オブジェクトは待ちの状態となる。また、オブジェクトの停止に対しても、他のオブジェクトは待ちとなる。

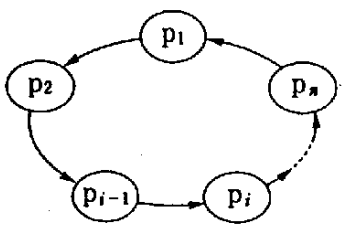


図4.8　トークンリング

4.3 デッドロック

4.3.1 通信デッドロックと資源デッドロック

デッドロックは、複数のオブジェクトが互いに他が獲得している資源を待ちあっている状態である [HOLT72]。例えば、オブジェクトAとBが資源xとyを利用しようとしているとする。Aがxの特権をもち獲得し、Bがyを獲得しているとする。このとき、AはBが獲得しているyを待ち、BはAが獲得しているxを待ち、互いに待ちの状態となる。これがデッドロックの例である。分散システムのデッドロックには、以下の二種がある。

(1) 通信デッドロック

(2) 資源デッドロック

A．通信デッドロック

AとBを二つのオブジェクトとする。AとBは、互いにメッセージの送信と受信を行おうとしているとする。Aは、メッセージの受信後にBに送信を行おうとし、Bも同様に、Aからのメッセージの受信後に送信を行おうとしている場合を考える。この場合、AとBは、ともに互いのメッセージ受信待ちとなってしまい、送信を行えなくなる。これを、通信デッドロックという。この原因としては、通信バッファが不足することが原因の一つとなる。

B．資源デッドロック

これに対して資源デッドロックは、複数のオブジェクトが資源を共有する場合に起きる。あるオブジェクトAが他のオブジェクトBの獲得する資源xを待ち、BがAの獲得する資源yを待つとき、AとBはデッドロックしている。ここでは、資源デッドロックについて考える。

4.3.2 デッドロックの定義

分散システムのデッドロック状態は、資源割当てグラフを用いて定義される。

[資源割当てグラフ]

(1) 各節点は、オブジェクトまたは資源を示す。

(2) オブジェクト p_i に資源 x が割り当てられているとき、節点 x から p_i に有向辺 x → p_i を設ける。p_j が x を要求しているとき、p_j から x に有向辺 p_j → x を設ける。

資源割り当てグラフにおいて、オブジェクト p_i から p_j に有向路があるときには、$p_i \Rightarrow p_j$ と書く。ここで、二つのオブジェクト p_i と p_j に対して、おのおの資源 y と x が割り当てられているとする。p_i と p_j がおのおの資源 x と y を要求し、オブジェクト p_k が y を要求するとする。このとき、図4.9に示す資源割当てグラフが得られる。ここで、$p_i \Rightarrow p_j \Rightarrow p_i$ であり、$p_k \Rightarrow p_j$ である。

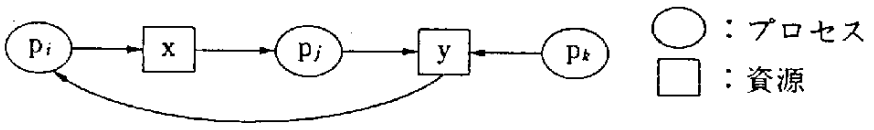


図4.9　資源割当てグラフ

以上の資源割当てグラフを用いて、デッドロック状態は以下のように定義される。

[デッドロック] 分散システムの状態を示す資源割当てグラフが巡回閉路を含むとき、システムはデッドロック状態である。

資源割当てグラフ内の巡回閉路内のオブジェクト p_i は、デッドロックになっている。即ち、$p_i \Rightarrow p_i$ である。図4.9にデッドロックの例を示す。図4.9に示す資源割当てグラフ内に、巡回閉路が存在する、即ち、$p_i \Rightarrow p_i$ で、$p_j \Rightarrow p_j$ であるので、オブジェクト p_i と p_j はデッドロックになっている。

4.3.3　資源獲得方法

各オブジェクトが、一度にいくつの資源を要求するかについて、以下のモデルがある[KNAP87]。

(1) 単一資源モデル。

(2) 積モデル。

(3) 和モデル。

(4) ('r) モデル。

単一資源モデルでは、オブジェクトは一時に高々一つの資源を要求する。資源割当てグラフで、各オブジェクト節点からの出力辺は高々一つである。通常のオブジェクトは、単一資源モデルである。

これに対して、積、和、('r)モデルでは、オブジェクトは一時に複数の資源x_1, ..., x_n ($n \geqq 1$) を要求する。積モデルでは、オブジェクトは、要求した全資源 x_1, ..., x_n が獲得されるまで実行を停止して待つ。従って、資源割当てグラフ内で各節点の出力辺は複数となる場合がある。また、デッドロックしているオブジェクトは複数の有向閉路に含まれ得るが、デッドロックの定義は単一資源モデルのものと同じである。即ち、資源割当てグラフ内の巡回閉路に含まれるオブジェクトがデッドロックしている。図4.10に、積モデルのオブジェクト p_i と p_j のデッドロックを示す。p_i は資源 x と y を要求し、p_j は a と b を要求するとする。また、p_i は資源 a をロックしていて p_j は x をロックしているとする。p_i と p_j は、巡回閉路に含まれるので、デッドロックしている。

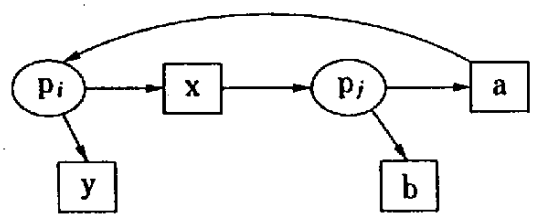


図4.10 積モデルオブジェクトのデッドロック

和モデルでは、オブジェクトは複数の資源 x_1, ..., x_n を要求するが、この中の一つでも獲得できれば実行を続けるものである。例えば、ホテルの予約を行う時に、幾つかのホテルに予約の要求を行って、どれか一つでも予約できればよい場合である。和モデルのオブジェクトのデッドロックの定義は少し複雑なものとなる。資源割当てグラフで有向閉路に含まれるオブジェクトがデッドロックしているとは限らない。各オブジェクト p_i は、複数の資源を要求しているので、p_i からのある資源への出力辺が巡回閉路に含まれても、他の資源が割り当てられれば、p_i は実行できるからである。図4.10に示した p_i と p_j を、和モデルのオブジェクトとする。p_i と p_j は巡回閉路内にあるが、デッドロックしていな

い。p_i は x を獲得できなくても、y を獲得できればよいからである。従って、和モデルのオブジェクトについてのデッドロックは、以下のように定義できる。

[結び] 資源割当てグラフ内の以下の条件を充足するオブジェクト節点の部分集合 K を結び (knot) とする。

(1) K 内の任意の二つのオブジェクト節点 p_j と p_k に対して、$p_j \Rightarrow p_k$ かつ $p_k \Rightarrow p_j$ である。即ち、結び K 内の任意の二つ節点について、一方から他方への有向路が存在する。

(2) K 内の節点だけが(1)の条件を満足する。

[和モデルのデッドロック] オブジェクト p_j が結びに含まれるとき、p_j はデッドロックしている。

図4.11に、3つのオブジェクト p_1、p_2、p_3 についての資源割当てグラフを示す。p_1、p_2、p_3 は、おのおの資源 x、y、z を獲得している。このとき、p_1 は z または y を要求し、p_2 は z または x を要求し、p_3 は x または y を要求する。このとき、p_1、p_2、p_3 は、結びに含まれるので、デッドロックしている。

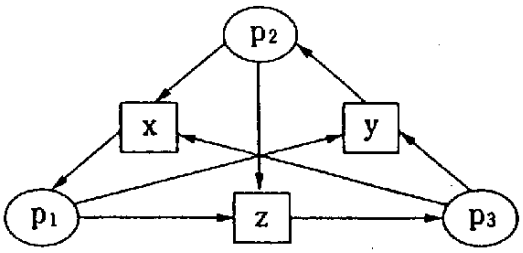


図4.11 結び

($^{\circ}$r) モデルは、積と和モデルを一般化したモデルである。オブジェクトは、n 個の資源を要求するが、このなかの r(≦ n) 個が獲得されたならば実行を続けるものである。単一資源モデルは ($^{1}_{1}$) モデル、積モデルは ($^{n}_{n}$) モデル、和モデルは ($^{n}_{1}$) モデルである。

4.3.4 デッドロック検出手順

デッドロックをどのように解決するかについて考える。デッドロックの解決方法として

は、以下がある。

(1) 検出 (detection) 方法。

(2) 防止 (prevention) 方法。

(3) 回避 (avoidance) 方法。

検出方法では、まずデッドロックを検出する。次に、デッドロック状態のオブジェクトの中から一つのオブジェクトを選択して、これを終了させる。防止方法では、資源を要求しているオブジェクトがデッドロックとならないように、資源を割り当てる方法である。回避方法では、デッドロックが生じる危険性があるとき、デッドロックが生じないようにある手段を講ずるものである。

A．検出方法

まず、検出方法について考える。

[デッドロック検出方法]

(1) 分散システム状態に対する資源割当てグラフ G を作り、巡回閉路を探す。

(2) 見つかれば、この巡回閉路内のあるオブジェクト p を選択し終了させる。

デッドロックの検出手順は、分散システム内の各オブジェクトがデッドロックしているかどうかを調べるものである。この検出手順は、以下の条件を満足するとき、正しい。

[検出手順の正しさ]

(1) すべてのデッドロックを有限時間で検出できる。

(2) 手順によりデッドロックを検出されたとき、分散システムはデッドロックしている。

デッドロック状態でなくても、デッドロック検出手順がデッドロック状態とが判断してしまう状態を偽デッドロック (false deadlock または phantom deadlock) という。例えば、図4.9で、オブジェクト p_i が資源 x を獲得していることを調べた後に、x を獲得しているオブジェクト p_j の状態を調べるとする。ここで、p_j は p_i が獲得している y を待っていることがわかり、デッドロックが起きていると判断する。しかし、p_i を調べている時間と p_j を調べている時間が異なっているので、p_i は、例えば x を獲得することを諦めて、他の資源を獲得しようとしているかもしれない。また、p_i は終了しているかもしれない。

このとき、デッドロックではない。このように、デッドロック状態でないのにデッドロックと判断してしまう状態が偽デッドロックである。分散システムは複数のコンピュータから構成されるために、同一時刻の各コンピュータの状態を獲得することができない。このために、偽デッドロックが生じる場合がある。

B．集中方式

集中方式では、デッドロック検出を行う指揮オブジェクト p_0 が存在する。p_0 は、分散システムの資源割当てグラフを持つ。

[集中処理方式1] [図4.12]

(1) 各オブジェクト p_i は、資源 x の獲得を要求するとき、指揮オブジェクト p_0 に要求を送信する。x を解放するときにも、解放要求を p_0 に送信する。

(2) p_0 は、x の獲得または解放要求を受けたならば、資源割当てグラフに変更を加える。獲得要求に対して資源割当てグラフを調べる。グラフが巡回閉路を含むならば、デッドロックが生じている。

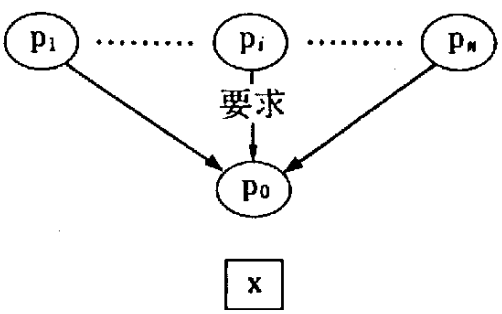


図4.12　集中処理方式

集中方式は、簡単であり、実装が容易であるが、以下の問題点がある。

[集中方式の問題点]

(1) 各オブジェクト p_i は、指揮オブジェクト p_0 と通信を行わねばならない。このために、p_0 に負荷が集中することになる。

(2) 指揮オブジェクト p_0 が障害を起こすと、他の全てのオブジェクトが待ちとなる。

他に、資源割当てグラフを、指揮オブジェクト p_0 に集める方法も考えられる。

[集中方式2]

(1) 各コンピュータ C_i は、C_i 内の資源オブジェクトについての資源割当てグラフ G_i を持つ。C_i は、周期的または指揮オブジェクト p_0 からの要求により、G_i を p_0 に送信する。

(2) p_0 は、G_1, ..., G_n から、分散システム全体の資源割当てグラフ G を作成する。G が巡回閉路を含むならば、デッドロックが起きていると判断する。

各コンピュータと指揮オブジェクト間の通信遅延がある。このために、指揮オブジェクト p_0 に資源割当てグラフ G_i が C_i から届いた時、G_i はその時点の C_i の状態を示していずに送信時の状態を示している。このために、この手順では擬デッドロックが生じる場合がある。各コンピュータの資源割当てグラフを同時に獲得できるならば、偽デッドロックは生じない。しかし、分散システムでは、同時に各コンピュータの状態を獲得することはできない。

C．分散方式

次に、指揮オブジェクトの存在しない場合について考える。オブジェクトが待ち状態になったときに、デッドロック検出手順が起動される。各コンピュータは、自分の資源についての資源割当てグラフを管理している。このグラフ内の情報が、デッドロック検出アルゴリズムの実行中に他のコンピュータに送信される。

まず、[CHAND83] の検出手順について述べる。この手順では、各オブジェクトは同時に複数の資源を要求できる積モデルである。各オブジェクトがある資源 x を要求し、待つときに検出手順が起動される。待ちとなったオブジェクトは、プローブ (probe) というメッセージを x を獲得しているオブジェクトに送信する。プローブは、三つ組 <i, j, k> として与えられる。ここで、i は待ちの状態となったオブジェクト p_i を示す。j はプローブを送信したオブジェクト p_j を示す。k は、プローブの宛先のオブジェクト p_k を示す。

[Chandy、 Misra、 Haas の手順]

(1) オブジェクト p_j が資源 x_j を獲得しているとする。ここで、他のオブジェクト p_i が

x_j を要求して待ちとなったとする。ここで p_i は、プローブ <i, i, j> を p_j に送信する。

(2) p_i が、プローブ <j, k, i> を受信したとする。j = i であるとき、p_i がはじめに送信したプローブが p_i に返ってきたことを示す。したがって、p_i はデッドロックしていることがわかる。

(3) (2)で、j ≠ i であるとする。p_i が要求しているがまだ獲得されていない各資源 x_h について考える。x_h は、オブジェクト p_h により獲得されているとする。p_i は、プローブ <j, i, h> を p_h に送信する。

[例] 図4.13に、5つのオブジェクト p_0、p_1、p_2、p_3、p_4 に例を示す。オブジェクト p_0 が資源 x を待ちとする。p_0 は、プローブ <0, 0, 1> を、x を獲得している p_1 に送信する。p_1 は、y と v を待っているとする。p_1 は、プローブを受信したならば、p_2 にプローブ <0, 1, 2> を送信する。また、p_4 に <0, 1, 4> を送信する。このようにして、p_1 は、p_3 からプローブ <0, 3, 1> を受信する。ここで、p_0 は、デッドロックしていることがわかる。

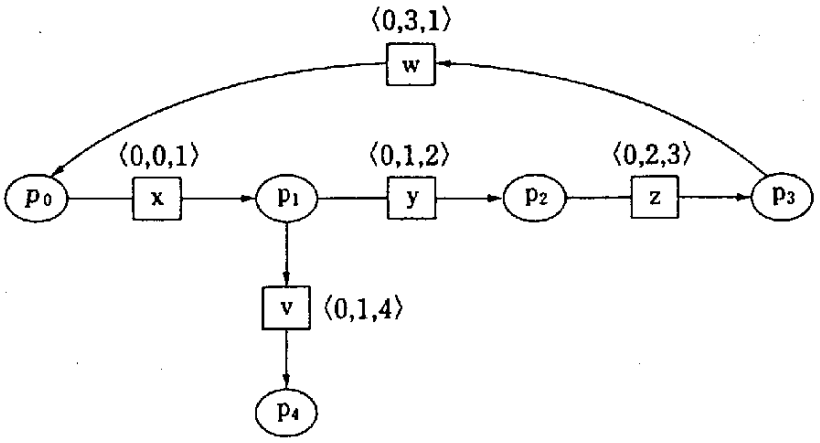


図4.13　デッドロック検出

4.4　合意

分散システムは、複数のオブジェクトから構成される。分散システム上の応用(分散アプリケーション)は、これらの複数のオブジェクトの協調動作により実現される。分散ア

プリケーションでは、複数のオブジェクト p_1, ..., p_n 間で種々の合意を行う必要がある。$n(\geqq 2)$ 個のオブジェクト p_1, ..., p_n の合意 (consensus) とは、各オブジェクト p_i がある値 v_i 持つことから出発して、オブジェクトが障害しても全オブジェクトが同一の値を決定することである。複数のオブジェクトが合意するためには、互いにメッセージの交換を行えばよいが、分散システムがどのような障害をこうむるかが問題となる。

4.4.1 障害

分散システムは、オブジェクトとオブジェクト間の通信を行うための通信ネットワークから構成される。したがって、障害には、オブジェクト障害と通信ネットワーク障害とがある。まず、オブジェクトの障害について考える。オブジェクトの障害には以下のものがある。

[オブジェクト障害]

(1) 停止障害。

(2) ビザンティン障害。

停止障害 (fail-stop) とは、オブジェクトが一定時間以上停止する障害である。停止した後、復旧しない障害を永久停止とする。停止障害には、間欠的に停止し直ちに復旧する障害は含まれない。

このような障害を、オミッション (omission) 障害という。オブジェクトが、送信されたメッセージに対して、応答を返したり返さなかったりする障害である。また、送信されたメッセージに対して、誤った応答を返す障害をコミッション (commision) 障害とする。オミッションとコミッション障害を、あわせてビザンティン (Byzantine) 障害 [LAMP82] とする。障害していないオブジェクトを、動作中または正しいオブジェクト (operational object) とする。

[ビザンティン障害]

(1) コミッション障害。

(2) オミッション障害。

分散システム内で、同時にどの位の数のオブジェクトが障害しているかも問題となる。

分散システムの障害として、以下の種類がある。

(1) 全体障害。

(2) 部分障害。

システム内の全てのオブジェクトが障害するとき、これを全体障害 (total failure) とする。また、一部のオブジェクトが動作しているとき、部分障害 (partial failure) とする。オブジェクト p_i が障害から復旧したとする。p_i の復旧後、どのように合意の処理を続けられるかにより、以下の二種の復旧がある。

[復旧方法]

(1) 独立復旧。

(2) 依存復旧。

オブジェクト p_i が、他のオブジェクトと通信を行わずに復旧できるとき、これを独立復旧 (independent recovery) とする。例えば、p_i が、他のオブジェクトと通信を行っていない場合には、独立復旧ができる。p_i が復旧したときに、他のオブジェクトと通信が必要であるとき、依存復旧とする。

4.4.2 合意プロトコル

合意プロトコルとは、複数のオブジェクト間での合意を行うためのプロトコルである。以下の条件を満足するオブジェクト $p_1,...,p_n$ 間のプロトコルを、合意プロトコル (agreement protocol) とする。

[合意プロトコルの条件]

(1) 全てのオブジェクト p_1, ..., p_n が同一の値に合意する。このとき、プロトコルが終了する。

(2) 各オブジェクト p_i は、プロトコルが実行される前にある値 v_i を持つ。これを p_i の入力値とする。入力値の集合を V とする。このとき、オブジェクト p_1, ..., p_n は、V 内のある入力値に最終的に合意する。

(3) 各オブジェクトは、有限の計算ステップで値を決定する。各オブジェクトの入力値の集合 $\{v_1,...,v_n\}$ を V とする。オブジェクト間の通信により、全オブジェクトが V 内のあ

る値 v_k を決定するためのプロトコルが、合意プロトコルである。例えば、旅行に行くかどうかを全員一致で決定することがある。このとき、各自の初期値は行くかどうかである。合意プロトコルは、各オブジェクト pi が取る初期値を交換する。交換された初期値に対して、p_i は新しい値を取る場合もある。このとき、新しく取られた値を交換しあう。この値交換を繰り返し行うことにより、最終的に一つの値に合意する。合意プロトコルの例として、分散データベースシステムで広く用いられている二相コミットメント (two-phase commitment) プロトコル [BERN87] がある。

4.4.3 停止障害があるときの合意プロトコル

合意プロトコルでは、システム内のオブジェクトの障害を考慮する必要がある。まず、オブジェクトの停止障害があるもとでの合意オブジェクトについて考える。

[停止オブジェクトの検出] 分散システムで、各オブジェクト p_i が、他のオブジェクト p_j が停止しているか動作中であるかは、p_j からメッセージが届くかどうかにより検出される。

一定時間以上の間、p_j から一つもメッセージを受信しないならば、p_i は、p_j が停止しているものと考える。

A．同期的オブジェクトと非同期的オブジェクト

他のオブジェクト p_j が障害しているかどうかの判断をオブジェクト p_i が行える前提には、p_jが、p_iが想定する速度以上で動作していることが前提となる。このとき、オブジェクト p_i と p_j は、同期的 (synchronous) であるとする。

p_j の処理速度が想定できないと、p_i は p_j からいつ応答が返ってくるかを決めることができない。同期的オブジェクトについて、停止障害が起きるもとで合意を行うプロトコルの例として、コミットメント制御がある。例えば、コンピュータが過負荷となり、p_j の速度が低下したことにより、p_i からのメッセージに対して一定時間以内に応答できないとする。p_i は p_j が動作しているにもかかわらず、停止しているものと判断してしまう。言い換えると、オブジェクトが同期的でない場合には、各オブジェクトは、他のオブジェクトの停止をタイムアウトにより検出できない。同期的でないオブジェクトを非同期的

(asynchronous)とする。非同期的オブジェクトに対して、以下の結論が示されている[FISC85]。

[非同期的オブジェクトの合意] オブジェクトが非同期的であるとき、一つでもオブジェクトが停止するならば、オブジェクトで合意を行えない。

4.4.4 ビザンティン合意

次に、オブジェクトがビザンティン障害を起こす場合を考える。すなわち、オブジェクトの障害について何も仮定できない場合である。オブジェクトがビザンティン障害を起こすもとで、複数のオブジェクト $p_1, ..., p_n$ が合意するとは何かを考える。

[ビザンチィン合意] ビザンティン(Byzantine)合意とは、オブジェクト障害について何も仮定できない場合に、正しい全オブジェクトが合意に達することをいう。ただし、同時に障害し得るオブジェクトの最大数はわかっているものとする。n個のオブジェクト $p_1,...,p_n$ の合意を考える。障害するオブジェクトの最大数をtとする。障害オブジェクトは、メッセージの送信をしなかったり、メッセージの中身を改変したり(嘘をつく)する。こうした障害オブジェクトが存在するもとで、あるオブジェクト p_i が送信したメッセージの中身(値)について、正しい全オブジェクトが同じ値に合意できることが問題となる。

[ビザンティン合意条件] 送信オブジェクト p_i がある値を持つメッセージmを、n-1個の受信オブジェクト $p_2, ..., p_n$ に送信したとき、以下の制約が満たされればビザンティン合意は達成される。

C1：$p_1, ..., p_n$ 内の全ての正しいオブジェクトは、同一の値に合意する。

C2：送信オブジェクト p_i が正しいならば、正しい全てのオブジェクトはmに合意する。

送信オブジェクト p_i が正しいとき、C2よりC1は明らかである。しかし、ビザンティン障害では障害について何も仮定できないので、p_i が正しいオブジェクトであるとは限らない。p_i が障害している場合には、正しい全受信オブジェクトは、何らかの値(mとは限らない)に合意することをC1は意味している。このようにオブジェクトの障害について何も仮定できない状況で、制約C1とC2を満たすプロトコルが必要となる。

送信オブジェクトp_iが障害している可能性があるので、受信オブジェクトp_iがメッセージmを受信したとき、他のp_jがどのようなメッセージをp_iから受信しているかを調べる必要がある。このために、受信オブジェクト同士が受信したメッセージを交換し合う。しかし、p_jが正しいことが保障されていない。受信したメッセージをオブジェクト間で交換しているときに、p_iとp_kから異なる値のメッセージを受信したとき、p_0とp_jのいずれかまたは両者のオブジェクトが障害しているのか区別できない。よって、メッセージをオブジェクト間で一回交換するだけでは、ビザンティン合意の決定を行えない。複数回のメッセージ交換が必要となる。

まず、通信ネットワークについて、以下の仮定を設ける。

[通信ネットワークについての仮定]

(1) 通信ネットワークは信頼性がある。送信したメッセージは、宛先に紛失されることも改変されることなく届く。

(2) 各オブジェクトから、全てのオブジェクトに通信路がある。どのオブジェクト間でも通信を行える。

(3) メッセージが到着しないことを、タイムアウトにより発見できる。即ち、通信ネットワークの遅延時間に上限がある。

このとき、以下の結果が得られている [LAMP82]。

[ビザンティン合意条件]

$n \geqq 3t+1$ のときのみ、ビザンティン合意を行える。

即ち、n個のオブジェクトのうち 1/3 以上が障害する場合には、正しいオブジェクト間で合意を行えない。

[例] 例として、3つのオブジェクトp_1、p_2、p_3 (n = 3) 間で合意を行うことを考える。ここで、障害オブジェクトの最大数を 1(t=1) とする。p_1を最初にメッセージを送信するオブジェクトとする。このとき以下を行う[図4.14]。

(1) 送信オブジェクトp_1が値aのメッセージを、受信オブジェクトp_2とp_3に送信する。

(2) 受信オブジェクトp_2とp_3は、おのおの受信したメッセージを交換する。

ここで、p_3が障害となっている。p_2は、メッセージaをp_1から、bをp_3から受信す

る。p_2は異なった値を受信するので、障害オブジェクトが存在することがわかるが、これがp_1なのかp_3なのかまたは両方なのかはわからない。さらに、オブジェクト間でメッセージの交換を行っても同じである。したがって、C2は満足されず、ビザンティン合意されない。

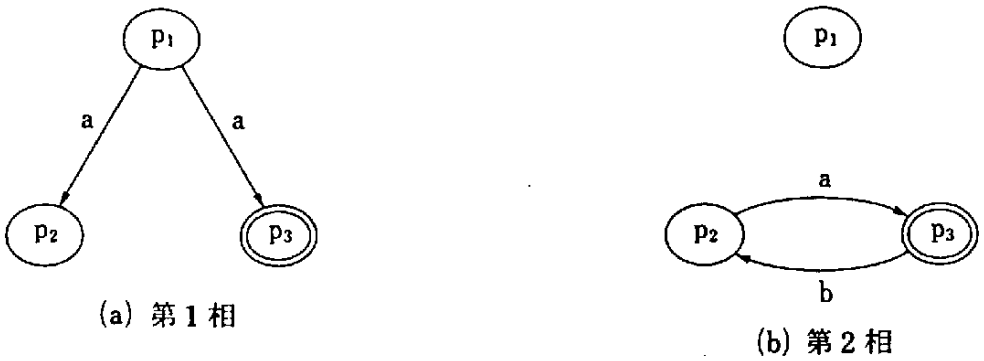


図4.14　p_3が障害している場合

次に、図4.14で送信オブジェクトp_1が障害している場合を考える[図4.15]。p_2は図4.14の例と同様の状況となる。よって、p_2とp_3は合意に達せず、C1は満たされない。

図4.14と図4.15の例が示すように、n＝3、t＝1では結論が出せない。

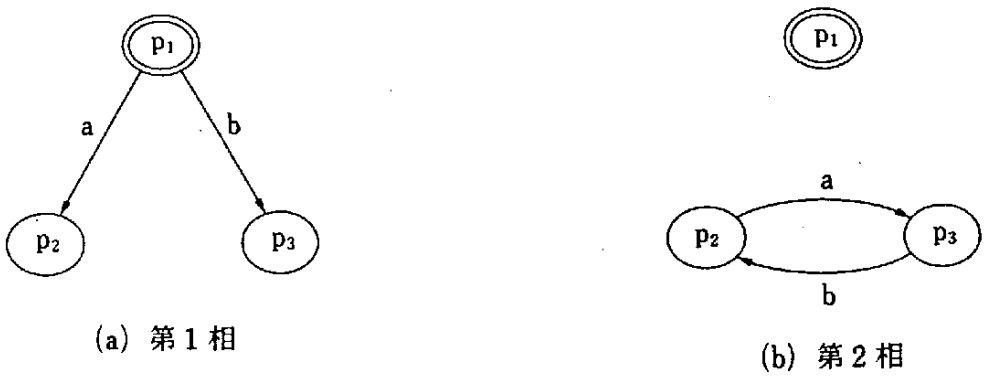


図4.15　p_1が障害している場合

ビザンティン合意プロトコルは、その実行ステップを示す相の系列から構成される。まず、第1相では、送信オブジェクトp_1が、n－1個の受信オブジェクト$p_2,\ldots,p_n$にメッセージを送信する。続く第2相では、受信オブジェクト同士が受信した値を交換し合う。例えば、p_2は受信したメッセージを、n－2個のオブジェクト$p_3,\ldots,p_n$に送信する。この

ときメッセージに、p_1の次にp_2によりこのメッセージが交換されてきたことを示す情報を付加する。各相で、オブジェクトp_iがメッセージmを受信したとき、mがこれまでにどのオブジェクトにより交換されたかがわかる。p_iは、mをまだ交換していないオブジェクトに送信する。p_iは、一定時間たってもp_jからメッセージを受信しない場合には、値としてv_0を受信したものとする。ここで、majorityを値の集合を引数とし、過半数の値を返す関数とする。例えば、majority ({a, b, a, a}) = aである。また、過半数を超える値がなければv_0を返す。以下に示す手順[図4.16]を行う。

[ビザンチィン合意手順]

(1) 送信オブジェクトp_1が、値vに自分の識別子を付けたメッセージ<v : p_1>を、n - 1個のオブジェクトp_2, ..., p_nに送信する。これを、第1相とする。

(2) 各受信オブジェクトp_iは、<v : p_1>をp_1から受信したとする(i = 1,...,n-1)。p_iは、メッセージ<v : p_1, p_i>をn - 2個のオブジェクトp_2, p_3, ..., p_{i-1}, p_{i+1}, ..., p_nに送信する。これを第2相とする。

(3) 各p_jは、第2相でメッセージ<v : p_1, p_i>をp_1とp_j以外のn - 2個のオブジェクトp_iから受信する。p_jは、メッセージ<v : p_1, p_i, p_j>を、p_1、p_i、p_j以外の残りのn - 3個のオブジェクトに送信する。これが第3相である。

(4) 以下同様に値の交換を行う。各h相で受信するメッセージは、h個のオブジェクトの識別子が含まれている。各オブジェクトp_iは、n - h個のオブジェクトにメッセージを送信する。

(5) t+1相で、各オブジェクトから得られたメッセージの過半数の値vを求める。これを合意値とする。

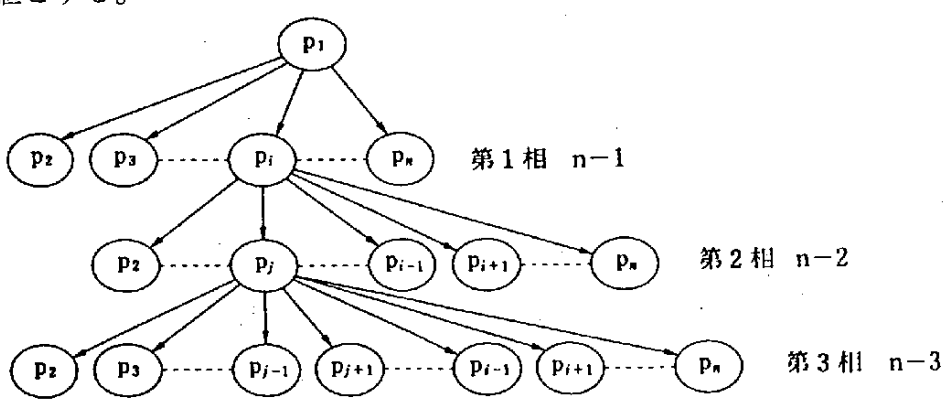
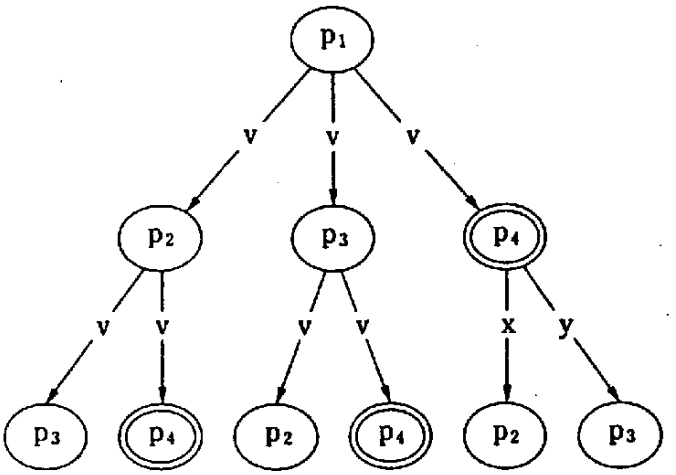


図4.16 メッセージ交換

[例] 例として、4つのオブジェクト p_1, p_2, p_3, p_4 間の合意について考える[図4.17]。ここで、オブジェクト p_4 が障害するとし、$t = 1$ とする。まず、送信オブジェクト p_1 がメッセージ v を送信する。正しい受信オブジェクト p_2 と p_3 は、v をおのおの p_3 と p_4、p_4 と p_1 に送信する。障害オブジェクト p_4 は、メッセージ x と y をおのおの p_2 と p_3 に送信する。ここで、p_2 と p_3 は、メッセージ v, v, x と v, v, y を持ち、v = majority(<v, v, x>) = majority(<v, v, y>) が過半数となり v に合意する。



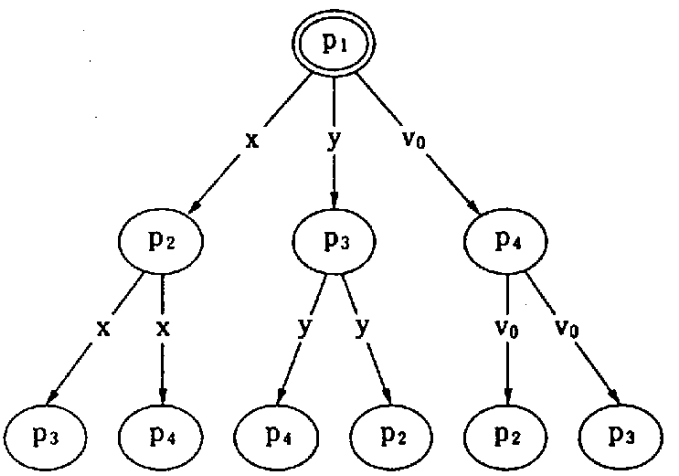
第1相：

第2相：

p_2 : v, v, x
p_3 : v, v, y

図4.17　p_4が障害オブジェクト

[例] 次に、図4.17で、送信オブジェクト p_1 が障害している場合を考える。図4.18に示すように、オブジェクト間で値を交換する。3つの正しいオブジェクト p_2、p_3、p_4 は同一の値の集合を受信し、同一のメッセージ x, y, v_{def} を得る。このとき、正しいオブジェクトは、majority 関数により、例えば、v_{def} に合意する。



第1相：

第2相：

p_2 : x, y, v_0
p_3 : x, y, v_0
p_4 : x, y, v_0

図4.18　p_1が障害オブジェクト

この例が示すように、ビザンティン合意では、正しいオブジェクトが何らかの同じ値を取る。取られた値が何かについては、議論していない。これは、応用により決まるものである。

[ビザンティン合意オブジェクトの性能]

次に、以上述べたビザンティン合意プロトコルの性能について考える。合意プロトコルの性能評価の尺度として、相数とメッセージの数と長さがある。

(1) 相数: 障害オブジェクトの最大数をtとしたとき、t + 1相の処理が必要である。これはビザンティン合意に至るための最小相数である。相数は、合意にいたるまでの時間を示しているので、合意に至る時間は、O(t) である。

(2) メッセージ数: 第k相では、(n - 1)(n - 2)...(n - k) 個のメッセージが送信される。従って、t + 1相で合意されるので、合意にいたるまでに送信されるメッセージ数は、(n - 1)(n - 2)...(n - (t + 1)) である。メッセージ数は、$O(n^{t+1})$ である。

(3) メッセージ長: 第k相で送信されるメッセージは、k個のオブジェクト識別子の系列を含んでいる。よって、t+1層で合意に達するので、メッセージ長はO(t) である。

以上から、ビザンティン合意プロトコルは、メッセージ数が $O(n^{t+1})$ となり、メッセージ数の点から現実のシステムには適用することは困難である。

4.4.5 署名を用いたビザンティン合意プロトコル

次に、各オブジェクト p_i がメッセージmを送信するときに、暗号化等によりディジタル署名 [DENN82] を行うことを考える。オブジェクト p_j がメッセージmを受信したとき、mの署名からmが p_i から送信されたことがわかる。また、p_j はmを改変して p_k に送信するとする。このとき、p_k はmが p_j により改変されたことがわかる。署名を用いることにより、障害オブジェクトにより値が改変されても、これを受信オブジェクトが発見できる。以下に、署名を用いた場合の合意プロトコル [LAMP82] の概要を示す。

[ビザンティン合意プロトコル]

(1) 送信オブジェクト p_i は、値vを含むメッセージmに署名して、受信オブジェクト

$p_2,\ldots,p_n$ に送信する。p_1 が障害していれば、各 p_j に、異なる任意の値を送信するかもしれないし、メッセージを送信しないかもしれない。

(2) 各 k 相で、各受信オブジェクト p_i は値 v を受信する。p_i は、v を含めたメッセージに自分の署名をして、まだそのメッセージに署名していないオブジェクトに送信する。これを k + 1 相とする。以下これを繰り返す。

(3) t + 1 相でメッセージの送受信は終了し、合意する値を決定する。こために、受信した値の集合に対して関数 choice を適用する。各オブジェクトは、同じ関数 choice を合意値を求めるために用いる。例えば、choice として署名無しのピザ合意プロトコルと同じく majority を用いることもできる。

第 k 相で、オブジェクト p_i は、p_j から値 v_j を受信したとする。次に、k+1 相で、p_i はオブジェクト p_k から値 v_k を受信したとする。ここで、p_k は、p_j から受信した値を v_k として p_i に送信したとする。v_j と v_k が異なっているとき、p_i は以下のいずれかが分かる。

(1) p_k が、p_j から受信した値を改変した。

(2) p_j が、p_i と p_k に異なった値を送信した。

従って、障害オブジェクトがメッセージを変更しても受信オブジェクトはこれを発見できる。例えば、図4.17で、正しいオブジェクト p_2 と p_3 は、p_4 からおのおの p_1 と p_3、p_1 と p_2 から受信した値 v と異なった値 x、y を受信する。従って、p_4 が障害していることが分かる。また、p_2 は p_3 から p_1 から受信した値と同じ v を受信しているので、p_3 が正しいことがわかる。p_3 も同様に、p_2 が正しいことがわかる。ここで、p_2 と p_3 は v に合意する。結果として、正しい全オブジェクト p_1、p_2、p_3 の間で、同一の値 v に合意できる。

以下の結論が得られている。

[署名付きビザンティン合意]

n ≧ t + 2 のとき、署名付きビザンティン合意プロトコルによりビザンティン合意が得られる。

例えば、3 つのオブジェクト (n = 3) 間で署名付きビザンティン合意オブジェクトを用いると、障害オブジェクトが 1 つ以下であれば、ビザンティン合意に達することができる。しかし、図4.17と図4.18に示したように、n = 3 で t = 1 の場合には、署名なしのプ

ロトコルでは、ビザンティン合意を行えない。

この手順では、署名なしの手順と同じく、t + 1 相で合意に達し、(n -1) (n -2)...(n - (t +1)) = $O(n^{t+1})$ のメッセージの交換がオブジェクト間で必要である。

4.4.6 放送型ネットワークを用いたビザンティン合意プロトコル

ここまで述べてきたプロトコルは、一対一型の通信ネットワークをオブジェクト間の通信に用いている。すなわち、n 個のオブジェクトにメッセージ m を送信しようとすると、オブジェクトは m を n 回送信する必要がある。これに対して、Ethernet、無線ネットワーク等の放送型の通信ネットワークを用いると、各オブジェクトはメッセージ m を一回送信すると、すべてのオブジェクトに m が送信される。従って、障害オブジェクトは、各オブジェクトに異なったメッセージを送信できない。通信ネットワークが、各オブジェクトで送信されたメッセージを、全オブジェクトに正しく届けれることが保障されている場合を考える。

[放送型ビザンティン合意プロトコル]

(1) 送信オブジェクト p_1 は、値 v を他の全オブジェクト $p_2,...,p_n$ に放送する。$v_i = v$ とする。これを第 1 相とする。

(2) v を受信したオブジェクト p_i は、v を他の全オブジェクトに放送する。

(3) p_i は、各オブジェクト p_j から受信した値を v_j に記憶する。すべてのオブジェクトから値を受信したならば、$<v_1, ..., v_n>$ を全オブジェクトに放送する。

(4) 全オブジェクトから受信した $<v_1, ..., v_n>$ の集合に対して、関数 choice を適用しその値を合意値とする。

以上の放送型ネットワークを用いたビザンティン合意プロトコルについては、以下の結果が得られている。

[放送型ビザンティン合意]

$n \geq t + 1$ ならば、ビザンティン合意できる。

4.5 後退復旧

分散システムのオブジェクトが障害をおこした場合の復旧について考える。

4.5.1 チェックポイント

分散システム内の複数のオブジェクト p_1, ..., p_n が協調して処理を行っているときに、あるオブジェクト p_i が障害を起こす場合がある。このとき、全てのオブジェクト $p_1,...,p_2$ が、始めから処理を実行し直すことも一つの方法であるが、各オブジェクトが同じ処理を再度行わねばならない。障害時に、各オブジェクトの処理時間を短縮するために、オブジェクトの始めから処理を行い直すのではなく、途中から処理を行う方法を考える。このためには、オブジェクトの実行途中の状態を記憶する必要がある。チェックポイント (checkpoint) は、オブジェクトの状態を記憶する時点である。ここでオブジェクト p_i が、現在の状態から過去の状態に戻ることを後退復旧とする。

[後退復旧 (rollback recovery)] オブジェクト p_i が、ある状態 s_i^1 から事象 $e_i^2,...,e_i^m$ により、状態 s_i^m に遷移したとする[図4.19]。 このとき、p_i の状態を s_i^m から、s_i^1 に戻すことを後退復旧とする。

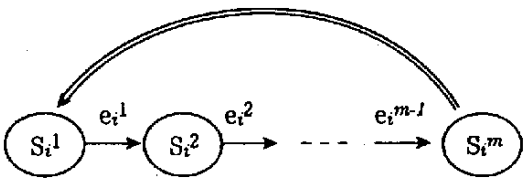


図4.19 後退復旧

オブジェクト p_i が状態 s_i^1 後退復旧するとき、事象 $e_i^2,...,e_i^m$ は起きなかったことになる。例えば、p_i がメッセージ m の送信を行っていれば、 m の送信はなかったことになる。このために、 m を受信したオブジェクトの受信も無効とする必要がある。このように、あるオブジェクトの後退復旧は、他のの後退復旧へと波及する場合がある。これを、連鎖的 (cascading) 後退という。

各オブジェクト p_i は、障害が起きてもデータが消失しない記憶システムを持つとし、これを安全記憶(ログ)とする。例えば、ディスク装置がこれにあたる。p_iは、状態s_i^kを安全記憶に記憶しておくと、障害が起きても安全記憶に格納した状態に戻ることができる。

[チェックポイント] オブジェクト p_i でチェックポイントを取るとは、p_i の状態を安全記憶に記憶することである。

オブジェクト p_i が障害を起こしたとき、最新のチェックポイントで記憶されている状態に p_i を後退復旧させ、ここから実行を再開できる[図4.20]。集中システムでは、こうした方法によりオブジェクトの障害に対して復旧を行っている。

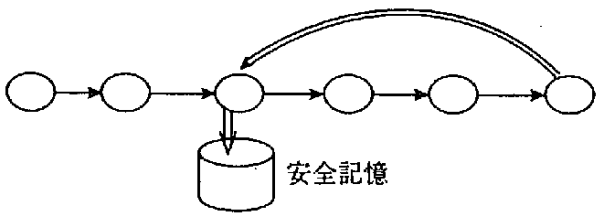


図4.20 チェックポイント

4.5.2 分散チェックポイントの問題点

分散システムでは、複数のオブジェクト $p_1,...,p_n$ が協調して処理を行っているために、オブジェクト間でどのようにチェックポイントを取り、再実行を開始するかが、問題となる。このとき、以下の問題点がある。

[分散チェックポイントの問題点]

(1) ドミノ効果。

(2) ライブロック。

図4.21では、二つのオブジェクト p_1 と p_2 が、チェックポイントを取りながら、互いに通信を行い処理を行っている。p_1 はチェックポイント cp_{12} を取った後に、メッセージ a

を送信し p_2 から b を受信している。この後に、チェックポイント cp_{11} を取りメッセージ c を送信している。p_2 は、a を受信した後に、チェックポイント cp_{21} を取り、メッセージ b の送信後に c を受信している。p_1 が障害を起こすと、最新のチェックポイント cp_{11} まで後退復旧する。p_1 は、cp_{11} 以降にメッセージ a を p_2 に送信しているので、p_2 も cp_{21} まで後退復旧する。p_2 は、cp_{21} の取得後にメッセージ b を p_1 に送信しているので、p_1 もさらに後退復旧する必要があり、cp_{12} まで後退復旧する。このように、あるオブジェクトの後退復旧により、次々に他のオブジェクトが後退復旧を繰り返す現象をドミノ効果 (domino effect) [RAND75]という。ドミノ効果を防ぐためには、各オブジェクトでのチェックポイントを、任意に取るのではなくある規則に従って他のオブジェクトと同期して取る必要がある。

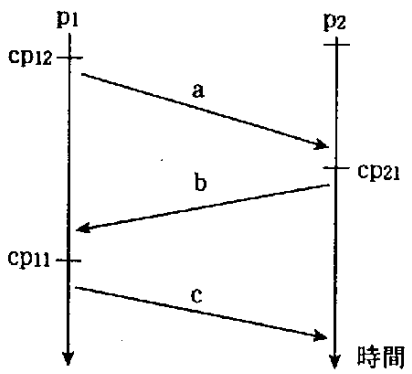


図4.21　ドミノ効果

次に、図4.22の例を考えてみる。オブジェクト p_1 は障害しチェックポイント cp_1 に後退復旧し、cp_1 から再実行したとする。p_1 は、メッセージ a を p_2 に送信してから、メッセージ b の受信を行い処理を続ける。ここで、p_1 が後退復旧することにより、a の送信が無効となるために、p_2 はチェックポイント cp_2 に後退復旧する。この後退復旧により b の送信が無効となる。これにより、p_1 は再度 cp_1 に後退復旧し、再実行されせる。p_2 も同様に、cp_2 に後退復旧し再実行される。このように、後退復旧と再実行が繰り返される現象を、ライブロック (livelock) という。ライブロックは、ここで示した例からわかるように、再実行を p_1 と p_2 が任意に開始すると起きる。

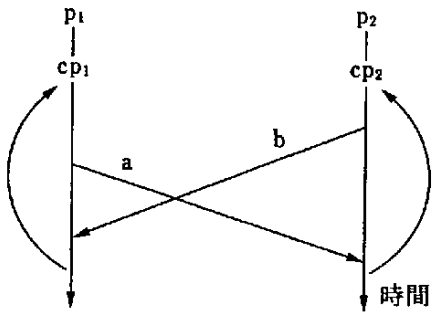


図4.22　ライブロック

以上示した例から、複数のオブジェクトが相互に通信を行いながら協調動作を行っている場合には、以下についての同期をとることが必要となる。

(1) オブジェクトのチェックポイント。

(2) オブジェクトの再実行の開始。

4.5.3　分散チェックポイント方式

分散システムでは、複数のオブジェクト p_1, ..., p_n 間でドミノ効果やライブロックが起こらないように、各オブジェクトでチェックポイントを取る必要がある。このために、チェックポイント間の関係について考える。ここで、cp_i はオブジェクト p_i のチェックポイントを示す。p_1, ..., p_nの全体チェックポイント(global checkpoint) cp とは、各オブジェクトのチェックポイントの組 $<cp_1, ..., cp_n>$ である。以前述べたカット (cut) を用いて無矛盾なチェックポイントを定義する。

[無矛盾なチェックポイント]

全体チェックポイントが無矛盾なカットであるとき、無矛盾である。即ち、あるオブジェクト p_i が、チェックポイント cp_i より以前に p_j よりメッセージ m を受信しているとする。このとき、p_j での m の送信も、チェックポイント cp_j より以前になければならない。図4.23は、3つのオブジェクト p_1、p_2、p_3 間のチェックポイントを示している。(1)は無矛盾なチェックポイントを示している。(2)のチェックポイントは矛盾している。

□：チェックポイント

図4.23 無矛盾なチェックポイント

[チェックポイントプロトコル]

無矛盾なチェックポイントを取る方法として、文献 [CHAND85] の方法を示す。各オブジェクト p_i に対して、以下を定義する。

$C_i = p_i$ が最後に取ったチェックポイント。

$Rec_i = C_i$ 以後に p_i が受信したメッセージを送信したオブジェクトの集合。

ここで、Rec_i 内のオブジェクトを p_i の子オブジェクト、p_i を親オブジェクトとする。チェックポイントの起動を行うオブジェクトを、根オブジェクトとする。

(1) チェックポイントを取ろうとする根オブジェクト p_i は、送信を停止して仮のチェックポイント TC_i をとる。Rec_i 内の全オブジェクトに $Checkp_i$ メッセージを送信する。

(2) オブジェクト p_j が $Checkp_i$ メッセージを受信したとする。ここで、p_j が仮チェックポイント TC_j をまだ取っていないならば、TC_j を取る。Rec_j 内の全オブジェクトに $Checkp_j$ メッセージを送信する。

(3) p_j が仮チェックポイントを既に取っているならば、Done メッセージを p_i に送信する。p_i が、p_j から Done を受信したならば、Rec_i から p_j を除く。p_i が Rec_i 内の全オブジェクトから Done を受信したならば、即ち、Rec_i が空となれば p_i を葉とマークして、OK メッセージを親オブジェクトに送信する。

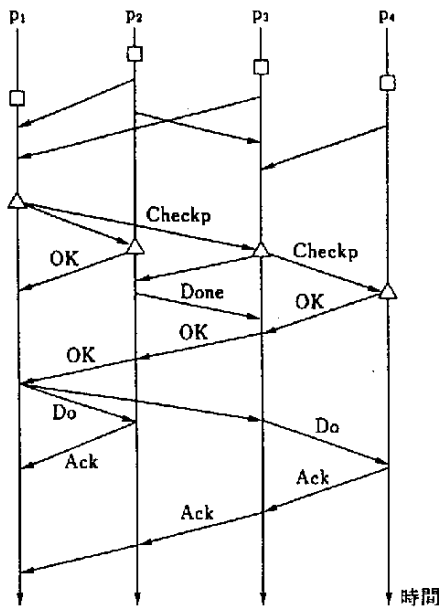
(4) p_i は、Rec_i 内の全オブジェクトから、OK を受信するまで待つ。Rec_i 内の全オブジェクトから OK を受信したならば、p_i の親オブジェクトに OK を送信する。

(5) p_i が根オブジェクトならば、仮チェックポイント TC_i をチェックポイント C_i とする。Rec_i 内の全オブジェクトに Do メッセージを送信する。

(6) p_i が Do を受信したならば、仮チェックポイント TC_i をチェックポイント C_i とする。p_i は Rec_i 内の全オブジェクトに、Do を送信する。p_i が葉ならば、Ack メッセージを親に送信する。

(7) p_i が、全ての子オブジェクトから、Ack メッセージを受信したならば、Ack を親に送信する。p_i が根オブジェクトならば、手順を終了する。

[例] 図4.24に示す4つのオブジェクト p_1, p_2, p_3, p_4 について考える。p_1 では、$Rec_1 = \{p_2, p_3\}$ である。オブジェクト p_1 は、チェックポイントを取った後にオブジェクト p_2 と p_3 からメッセージを受信している。また、p_3 は、p_2 と p_4 からメッセージを受信している。p_1 が仮チェックポイントを取る。このとき、$Rec_1 = \{p_2, p_3\}$ である。p_1 は、Checkp メッセージを p_2 と p_3 に送信し、p_2 と p_3 は仮チェックポイントを取る。$Rec_2 = \phi$ であるので p_2 は、仮チェックポイントを取った後に OK メッセージを p_1 に返す。$Rec_3 = \{p_2, p_4\}$ であるので、p_3 は Checkp メッセージを p_2 と p_4 に送信する。p_2 は既に仮チェックポイントを取っているので、p_3 に Done メッセージを送信する。p_3 は、$Rec_3 = \{p_4\}$ とする。p_4 は、仮チェックポイントを取り、Done メッセージを p_3 に返す。ここで、p_3 は Done を p_1 に送信する。p_1 は、全オブジェクトで仮チェックポイントが取られたことがわかるので、p_2 と p_3 に仮チェックポイントをチェックポイントとするための DO メッセージを送信する。p_2 と p_3 は仮チェックポイントをチェックポイントとする。ついで、p_3 は p_4 に同様に DO メッセージを送信する。



△：仮チェックポイント　□：チェックポイント

図4.24　分散チェックポイント

以上の手順により、複数のオブジェクト間で無矛盾なチェックポイントを取ることができる。

4.5.4 復旧方式

あるオブジェクト p_i が障害(停止)した場合に、チェックポイントを用いてどのように復旧するかを考える。ここで、各オブジェクト p_i に対して、以下を定義する。

Snd_i = 最新のチェックポイント C_i 以後に、p_i がメッセージを送信したオブジェクトの集合。

ここで、Sndi内のオブジェクトをp_iの子オブジェクトとする。また、p_iを親オブジェクトとする。障害したオブジェクトを根オブジェクトとする。

[復旧方式]

(1) p_i は、チェックポイント C_i に後退復旧する。p_i は、Recovery メッセージを Snd_i 内の全オブジェクトに送信する。

(2) p_j が Recovery を受信したとする。p_j がまだ後退復旧していなければ、C_j に後退復旧する。p_j は、Recovery メッセージを Snd_j 内の全オブジェクトに送信する。p_j が既に後退復旧していれば、Done メッセージを親オブジェクトに返す。

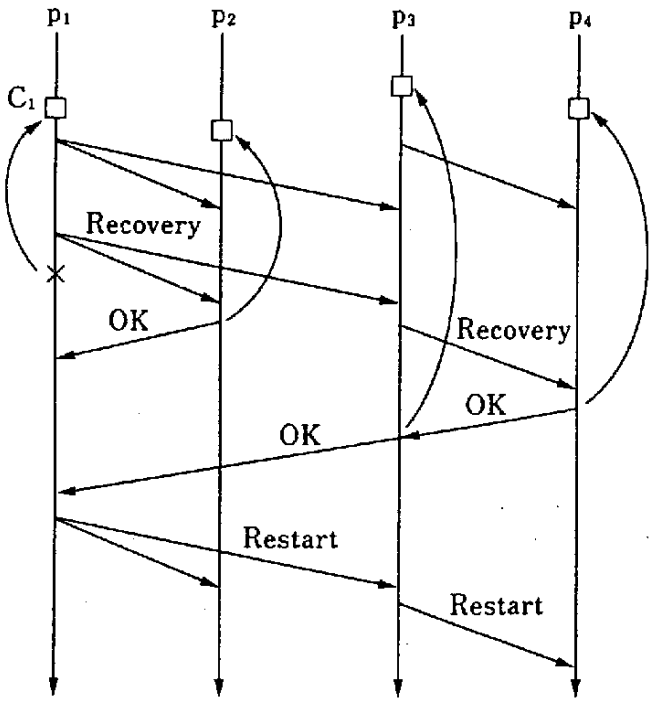
(3) p_j が p_k から Done を受信したならば、Snd_i から p_k を除く。全ての子オブジェクトから Done を受信したならば、即ち、Snd_j が空となれば、p_j を葉とマークする。親オブジェクトに、OK メッセージを送信する。

(4) 全ての子オブジェクトから、OK または Done を受信したならば、親オブジェクトに OK を送信する。p_i が根オブジェクトならば、Restart メッセージを Snd_i 内の全オブジェクトに送信する。

(5) p_j が、Restart メッセージを受信したならば、Snd_i 内の全オブジェクトに Restart メッセージを送信する。p_j が、葉オブジェクトならば、再実行を開始し Ack を親オブジェクトに送信する。すべての子オブジェクトから Ack メッセージを受信するならば再実行を開始し、Ack を親オブジェクトに送信する。p_j が、根オブジェクトならばアルゴリズムを終了する。

[例] 図4.25を考える。オブジェクト p_1 が障害し、最新のチェックポイント C_1 まで後退

復旧する。ここで、p_1はチェックポイント後にオブジェクトp_2とp_3にメッセージを送信しているので、p_2とp_3も後退復旧する必要がある。即ち、$Snd_1 = \{p_2, p_3\}$であるので、p_1はRecoveryメッセージをp_2とp_3に送信する。p_3はp_4にメッセージを送信している、即ち、$Snd_3 = \{p_4\}$である。p_3はRecoveryメッセージをp_4に送信する。p_4は、Recoveryメッセージを受信すると、チェックポイントC_4に後退復旧しDoneをp_3に返す。p_2とp_3もチェックポイントに後退復旧し、Doneメッセージをp_1に返す。p_1はp_2とp_3からDoneメッセージを受信したならば、Resatrtメッセージをp_2とp_3に送信する。p_3はRestartをp_4に送信する。ここで、全オブジェクトが処理を再開する。



□：チェックポイント

図4.25 チェックポイントと復旧

4.6 フォールトトレランス

分散システムでは、システムがこうむる種々の障害に対して、頑強であることが求められる。この問題について考える。

4.6.1 障害

複数のオブジェクトから構成される分散システムは、利用者に、ある仕様を満足するサービスを提供するシステムである。このとき、システムが、仕様を満足するサービスを提供できなくなったとき、システムは故障 (failure) したという。システムの障害をもたらすシステム要素の障害を障害 (fault) という。システム内の障害によって、利用者からみて正しくない状態が、誤り (error) である。障害が生じても、システムを障害せずに、サービスを提供できるシステムをフォールトトレラント (fault tolerant) であるという。

フォールトトレランスとするための一つの方法として、システムの構成要素を多重化することがある。これに対して、フォールト回避 (fault avoidance) とは、高信頼な要素からシステムを構成することにより、フォールトが起きないようにすることである。

これまで、システムのフォールトに対する頑強性の尺度として、信頼性、可用性が用いられてきた。システムの信頼性 (reliability) とは、フォールトに対してどの程度システムが頑強であるかを示している。一般に、平均故障間隔 (MTBF : Mean Time Between Failure) が長いほど、システムの信頼性は高いといえる。一方、システムの可用性 (availability) は、どの位直ちにシステムを利用できるかを示している。MTBF が長いことに加えて、平均修復時間 (MTTR : Mean Time To Repair) が短い程、システムの可用性は高い。この他に、安心性 (safety)、安全性 (security) といった尺度もある。安心性は、システムがカタストロフィをどの位起こさないかを示している。安全性は、利用権のない利用者が不正に情報を利用することを、どの位防止できるかを示している。

信頼性、可用性、安心性、安全性の概念を包含した概念として、ディペンダビリティ (dependability) が示されてきている。ディペンダビリティとは、システムが利用者に対するサービスをどの位仕様通りに提供するかを示した概念である。

[フォールトトレランス機能]

フォールトトレランスは、障害が起きたとしても仕様を満足するサービスを提供する機能である。これは、以下の二つの機能から構成される。

(1) 誤り処理 : 誤りを除去することである。これには、誤りを検出し、除去する機能からなる。

(2) フォールト処理 : 障害が、さらに誤りを起こさせないようにする機能である。これ

には、障害診断 (diagnosis)、不活性化 (passivation) がある。診断は、観察された誤りの原因を見つけることである。不活性化は、診断された障害が再度起きることを防止することである。

4.6.2 フォールトトレランス技術

分散システムをフォールトトレランスにするための方法として、システムの構成要素を多重化することがある。要素 (オブジェクト) を多重化したものを、要素の複製 (replica) とする。一つの複製が障害しても、残りの複製によりサービスを提供することにより、システムをフォールトトレランスとするものである。

分散システム内のオブジェクト p_i の複製 (replica) を p_{i1}, ..., p_{in} とする。各複製 p_{ij} は、おのおの異なったコンピュータで処理されるとする。ここで、コンピュータの障害に対して、オブジェクト p の処理を継続することを考える。ここで、以下の仮定を設ける。

[仮定] 各コンピュータの障害は、他のコンピュータと独立に起きる。以上の仮定を満足する障害を、ハイゼンバグ (heisenbug) という。例えば、一つのコンピュータにより複数のコンピュータが集中管理されていると、このコンピュータが故障するとシステム全体が動作しなくなってしまうが。これは、ハイゼンバグではない。

[複製方法]

オブジェクト p_i から、複数の複製 p_{i1}, ..., p_{in} を作成する方法として以下がある。

(1) 能動的複製化 (active replication)。

(2) 受動的複製化 (passive replication)。

(3) 準能動的複製化 (semi-active replication)。

能動的複製 [ACHN93] について考える。どの複製 p_{ij} も、全く同じに実行される。即ち、各 p_{ij} は、同じ入力を得て、同じ計算を行い、同じ結果を出力する。このために、オブジェクトは決定論的である必要がある。複製がビザンティン障害を起こしても、各複製からの出力を比較し、多数決を取ることにより、正しい出力を決定できるとともに障害複製を検出できる。

次に、受動的複製 [BUDH94] について考える。複製の中の一つ、例えば、p_{i1} を主複製とし他を従複製とする。主複製 p_{i1} は、入力を得て計算を行う。他の従複製 p_{ij} は、計算を

行わない。p_{i1} は、周期的にチェックポイントを取り、このときの p_{i1} の状態 s_{i1} を全従複製 p_{i2}, ..., p_{in} に送信する。各 p_{ij} は、p_{i1} から状態 s_{i1} を受信したならば、p_{ij} の状態を s_{i1} とする。ここで、p_{i1} でのみ計算が行われ、他の従複製は p_{i1} の計算に従うだけであるので、オブジェクト p_i は決定論的である必要はない。p_{i1} が障害した場合には、従複製の中の一つ p_{ik} が主となり、計算をつづげる。このとき、p_{ik} は最新のチェックポイントから計算を再開する必要がある。

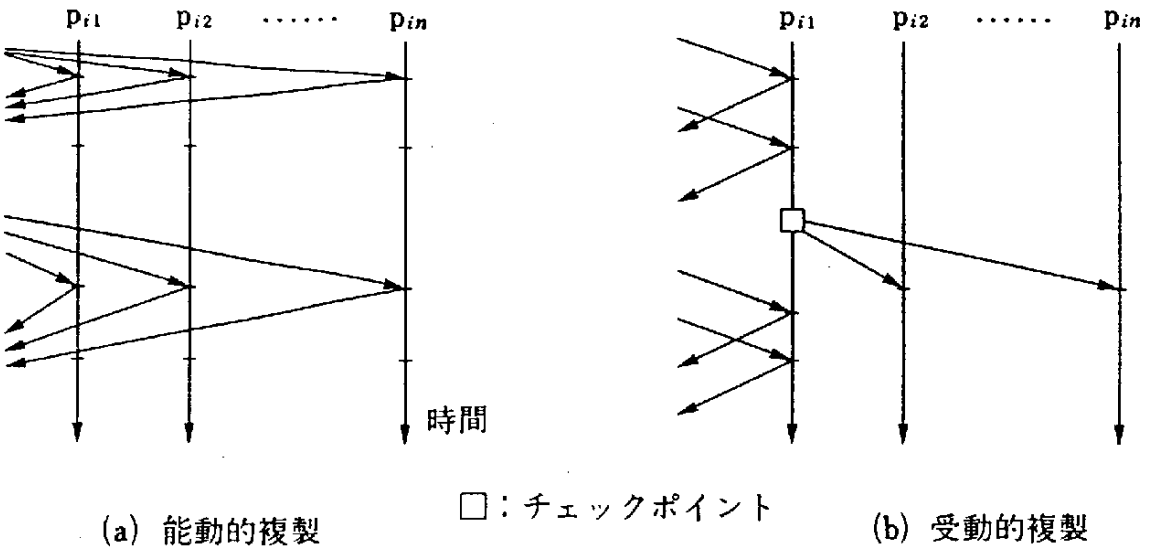
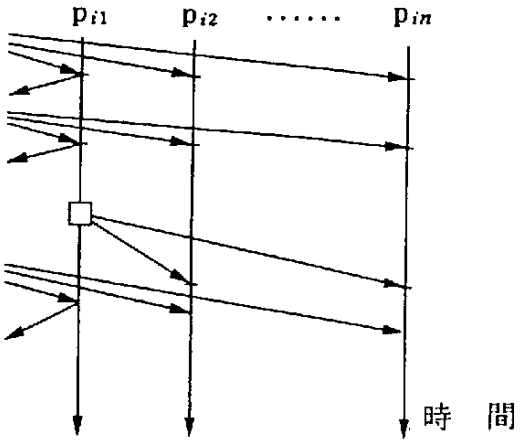


図4.26　能動的と受動的複製化

最後の準能動的複製 [THOMA90] は、能動的複製と受動的複製を複合したものである。能動的複製では、各複製が同一の計算を行っているので、どのコンピュータが障害しても、他の複製はサービスをそのまま継続できる。しかし、オブジェクトが決定論でなければならない。これに対して、受動的複製では、オブジェクトは非決定論的でもよいが、主複製 p_{i1} が停止した場合には、他の複製 p_{ik} がチェックポイントから計算をやり直さねばならない。非決定論的なオブジェクトを、コンピュータの障害に対してサービスを影響なく継続できることを考える。準能動的複製では、主複製 p_{i1} が存在する。各複製は、同一の入力を受け取り、計算を行うが出力は行わない。p_{i1} だけが計算結果の出力を行う。受動的複製化と同じく、p_{i1} はチェックポイントを取り他の複製 p_{ij} に送信する。p_{ij} は、チェックポイントを受信すると自分の状態を p_{i1} と同じにする。例えば、p_{i1} は非決定論的な分岐があるときに、チェックポイントを取り、状態を従複製に送信する。このように、他の複製は計算を p_{i1} と同じく行うことになる。



□：チェックポイント

図4.27　準能動的複製化

5章　安全性

分散システムでは、不正利用者が不正な方法で、システム内の資源を操作することを防止する必要がある。このための制御が、安全性 (security) 制御である。

5.1　安全性とは

まず、分散システムの安全性とは、何かについて考える。

[安全性] 不正利用者により、分散システム内の資源 (データベース、プロセス、利用者等) が不正利用されることが防がれているとき、分散システムは安全 (secure) であるという [図5.1]。

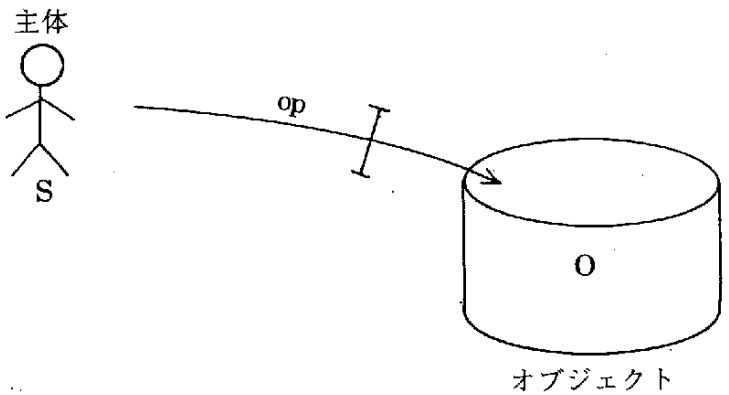


図5.1　安全性

インテグリティ制約とは、分散システム内の資源が保つべき条件である。これに対して、安全性は利用者がどのようにシステムを利用するかに関している。

分散システムの安全性を考えるとき、どのような利用者がシステム内のどの資源をどのように利用できるかが重要となる。誰がどの資源をどのように利用できるかを示す規則をアクセス規則 (access rule) という。アクセス規則に適合しない利用者が分散システムを利用するならば、安全性が乱されることになる。誰が、どの資源をどのように利用 (例、read,　write) するかのアクセス規則を与えることを権限付与 (authorization) という。権

限付与を行う利用者を、権限付与者 (authorizer) という。

分散システムの安全性を保つためには、以下の 3 つの制御が必要である。

(1) アクセス制御。

(2) 情報流制御。

(3) 推論制御。

アクセス制御 (access control) とは、システム内の資源に対するアクセスが権限付与された通りに行われていることを保証するための制御である。システム内の資源を利用しようとする利用者等の実体と、利用されるデータベース等の資源間の関係を、アクセス規則が保たれているように制御することである。

次に、情報流について考える。例えば、情報検索システムについてである。情報検索システムから検索したデータを自分のパソコン内のディスクにダウンロードし、その複写を情報システムの利用資格のない利用者に渡すことは不正な情報流である。こうした不正な情報流を制御することが情報流 (information flow) 制御である。

推論制御とは、保護されているデータを公開されているデータから推論されることを防止するための制御である。例えば、住民についてのデータベースシステムで、各個人の情報は秘密にされていても、住民について公表されている平均年令等の統計データをもとに、ある個人の年令等の情報を推論されることを防止するための制御である。

5.2 アクセス制御

まず、アクセス制御について考える。アクセス制御では、誰が、どの資源をどのように利用できるかというアクセス規則が重要となる。

5.2.1 システムの構成

ここで、分散システムは、オブジェクト (object) と主体 (subject) から構成されると考える。

(1) オブジェクト。

(2) 主体。

オブジェクトとは、ある操作演算を実行する実体である。オブジェクトは、演算(メソッド)を受けるとこの演算を実行し結果を返す。これに対して主体とは、オブジェクトに対して操作演算の実行を指示する実体である。主体とオブジェクトの関係は相対的である。例えば、プログラムとファイルの間では、プログラムが主体でファイルはオブジェクトであるが、利用者とプログラムの関係では利用者が主体でプログラムはオブジェクトとなる。ここで、主体がオブジェクトに対して、どのような操作演算を行えるかが問題となる[図5.2]。

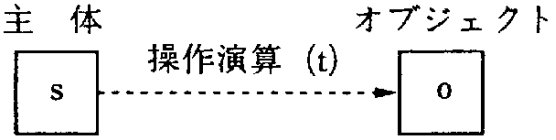


図5.2　アクセス規則 <s, o, t>

分散システム内の主体の集合をS、オブジェクトの集合をO、アクセス型(操作演算の種類)の集合Tとする。このとき、アクセス規則は、以下のように定義される。

[アクセス規則] アクセス規則集合Rは、R　S × O × Tであり、各組 <s, o, t> ∈ A をアクセス規則とする。

アクセス規則集合R内のアクセス規則 <s, o, t> は、誰(s)が、何(o)を、どのように(t)にアクセスしてよいかを示している。例えば、Aさんがファイルオブジェクト F を read と write できるときには、アクセス規則は、<A, F, read> と <A, F, write> となる。一方、Bさんが F を read のみできる場合には、<B, F, read> となる。Bさんが F を read しようとする要求も、アクセス規則と同じ形式で <B, F, read> と書ける。これは、アクセス規則を満足するので受け付けられ、Fがreadされる。しかし、要求 <B, F, write> は、アクセス規則にないので受け付けられない。

アクセス規則集合Rは、二次元のマトリクスAとして表すことができる。各主体sとオブジェクトoに対して、<s, o, t> ∈ Rであるとき、Aのs行o列A[s, o]は、tとなる。このマトリクスAをアクセスマトリクスとする。

[例] アクセスマトリクスの例を図5.3に示す。主体として人事部長と秘書があり、オブ

ジェクトとしてデータename、eno、salがある場合を示している。人事部長は全データに対して、全ての演算を行うことができる。これに対して、秘書は、データenameとenoをreadできるだけである。

	ename	eno	sal
人事部長	all	all	all
秘書	read	read	——

図5.3　アクセスマトリクス

主体からのアクセス要求も、アクセス規則と同一の形式<s, o, t>で記述される。アクセス要求<s, o, t>に対して、アクセス規則集合Rが調べられて、条件を満足すればこのアクセス要求は受け付けられる。そうでなければ拒否される。

アクセス規則集合Rをアクセスマトリクスとして実現することは、マトリクスの記憶量の点から困難となる。このために、アクセスマトリクスのの実現方法として以下の二つがある。

(1) アクセスリスト (access list) 方法。

(2) ケーパビリティリスト (capability list) 方法。

A．アクセスリスト

アクセスリスト方法をまず、考える。各オブジェクトoに対するアクセスリストAL(o)はオブジェクトoを利用できる主体sが何であり、それが許されるアクセス型がtであることを示している。即ち、

$$AL(o) = \{ <s, t> | <s, o, t \in R\}$$

利用者からのアクセス要求<s, o, t>を受けた時、まず、この利用者がsであることを確認せねばならない。これを認証 (authenitication) といい、利用者からのパスワード等により確認を行う。次に、オブジェクトoのアクセスリストAL(o)を調べ、<s,t>がAL(o)内に存在することを確認する。この方式は、現在のコンピュータのアクセス制御方法として用いられているものである。オブジェクト毎にアクセスリストが管理され、利用

者からのアクセスが行われる毎に、アクセスリストが調べられるものである。

B．ケーパビリティリスト

これに対して、ケーパビィリティリスト (capability list) 方法では、各主体 s に対して s が利用できるオブジェクト o と、o に対するアクセス型 t が <o, t> として与えられている。即ち、主体 s に対するケーパビィリティリスト CL(s) は、以下のように定義される。

$$CL(s) = \{ <o, t> | <s, o, t> \in R\}$$

ここでは、<o, t> を s のケーパビィリティ (capability) という。ケーパビィリティを持っている主体だけがオブジェクト o をアクセスできる。従って、システムは、各アクセス要求に対して、誰 (s) が何 (o) をどの (t) のようにアクセスするかを調べる必要がない。しかし、アクセス規則の変更が生じた場合には、利用者に分散しているケーパビリティの変更を行う必要がある。このために、利用者の数が大きなシステムでは有効なものとなるが、現在のシステムでは用いられていない。

ケーパビリティリスト方式は、ケーパビリティという切符を持った利用者だけがデータベースシステムを利用でき、システムは切符が正しいものかどうかをチェックすればよい。これに対してアクセスリスト方式では、座席予約と同じくアクセス要求を出した利用者が既に予約登録されているかどうかが予約表 (アクセスリスト) でチェックされる。利用者毎に表のチェックが行われ、システムとして表を保持していなければならない負荷があるが、ケーパビリティリスト方式ではこの負荷がない。

C．アクセス制御の問題点

アクセス制御方式の問題点としては、不正な情報流を制御出来ない点である。例で考える。図5.4で、主体 A はファイル F と G を read と write できるとする。一方、主体 B は F を read 出来ないが、G を read できるとする。A がファイル G からレコード x を読みだし、F に書いたとする。その後、B が F 内に書かれたレコード x を読みだしたとすると、結果的に G 内の x を読んだことになってしまう。このように、B はファイル G をアクセスできないが、A とファイル F を経由して結果として G 内の情報が B に不正に流れてしまう。アクセス制御では、オブジェクトと主体間で各主体がどのオブジェクトをどのように操作できるかが制御されるが、こうしたファイル G から主体 B への不正な情報流

を制御出来ない。

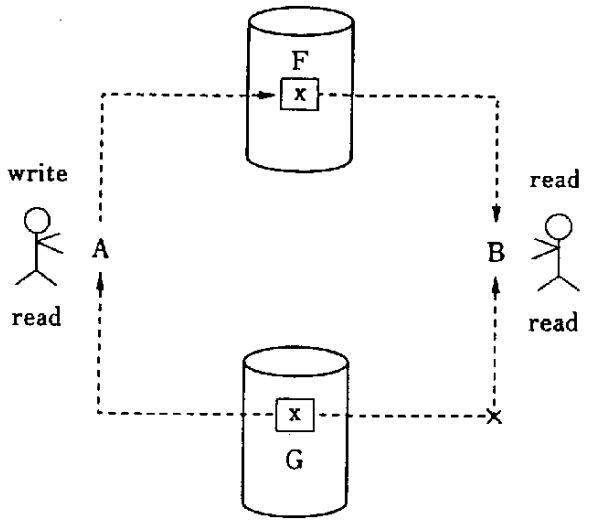


図5.4　不正な情報流

5.3　情報流制御

分散システムで、情報流を制御する問題を考える。

5.3.1　情報流

情報流を制御するために、抽象代数の束 (lattice) に基づいたモデルが、Dening 夫妻等 [DENN82] により示されている。システム内の利用者 (subject)、資源 (object) といった要素は、ある安全性クラス (security　class) に属している。システムの安全性のクラスの集合を S とする。システム内の二つの要素を e_1 と e_2 とする。各々の安全性クラスを class (e_1) と class (e_2) とする。e_1 の持つ情報を e_2 に渡してよいとき、以下のように書く。

$$\text{class}(e_1) \rightarrow \text{class}(e_2)$$

ここで、→は、S 上の半順序関係であり (→⊆ S × S)、流出可能 (can-flow) 関係である。

[例] 例えば、図5.4の例で、ファイル F と G の安全性クラスを f と g とし、利用者 A と B の安全性クラスを a と b とする。ここで、G → a → F とし、F →a も a → G も

成りたたないとする。即ち、利用者Aは、G内のデータxをファイルFに渡すことができる。しかし、この逆は行えない。ここでF → bとする。G → a → Fであるので、結果としてG → bとなり、bはG内のデータを読めることになる。従って、このことを防止するためには、F → bであってはならないことになる。

5.3.2 束モデル

以上の要素間の情報流の関係は、束 <S, →, ∪, ∩ > として表せる。ここで、∪は最小上界を示し、∩は最大下界を示す。S内の任意のクラス s_1 と s_2 に対して、$s_1 \cup s_2$ はS内のクラスsであり、$s_1 \rightarrow s$、$s_2 \rightarrow s$であり、かつ $s_1 \rightarrow s_3$、$s_2 \rightarrow s_3$、$s_3 \rightarrow s$ なるクラス s_3 が存在しないものである。$s_1 \cap s_2$ も同様に定義される。

任意の安全性クラス s_1 と s_2 の間に、以下の支配関係 (dominant relation) を定義する。

[支配関係]

(1) $s_1 \rightarrow s_2$ で、$s_2 \rightarrow s_1$ でないならば、$s_1 > s_2$ である。

(2) $s_1 > s_2$ または $s_1 = s_2$ ならば、$s_1 \geqq s_2$ である。このとき、s_1 は s_2 を支配する (dominate) とする。

(3) $s_1 \geqq s_2$ または $s_2 \geqq s_1$ ならば、s_1 と s_2 は比較可能である。

次に、情報流を満足するように、アクセス規則を与えることを考える。このようなアクセス制御を強制アクセス制御 (mandatory access control) という。文献 [BELL73] によるモデル (BLP(Bell-LaPadula) モデルという) を述べる。ここで、sをシステム内の主体、oをオブジェクトとする。主体sがオブジェクトoを操作するとする。

[BLP モデル]

(1) 単純規則 : class(s) ≧ class(o) ならば、sはoを読める。

(2) * 規則 : class(s) ≦ class(o) ならば、sはoに書き込める。

[例] 例として、会社の情報について考える。会社には、開発部 (D)、営業部 (S)、総務部 (A) があるとする。また、情報には、最高機密 (t : top secret)、重要機密 (s : secret)、機密 (c : confidential)、公開 (u : unclassified) の種類があるとする。このとき、会社のデータと利用者を、組織の種類 (D、S、A) と安全性のレベル (t、s、c、u) によ

り分類する。会社内の各実体 e に対して、安全性クラス class(e) を、組織の種類 $C \subseteq \{D, S, A\}$ と安全性レベル $L \in \{t, s, c, u\}$ の組 <C, L> により示す。 ここで、安全性レベル間で t は s よりもレベルが高く、s は c よりもレベルが高く、c は u よりもレベルが高い (t ＞ s ＞ c ＞ u)。例えば、会社の社長 p は、class(p) = < {D, S, A}, t> である。ここで、任意の安全性クラス $s_1 = <C_1, L_1>$ と $s_2 = <C_2, L_2>$ に対して、以下の関係を定義する。

$C_1 \supseteq C_2$ でかつ、$L_1 \geqq L_2$ ならば、 class $(s_1) \geqq$ class (s_2) である。ここで、二つのクラス $L_1 = <\{D, S\}, t>$ と $L_2 = <\{D, S\}, s>$ に対して、$L_1 > L_2$ である。

s を利用者、o をデータとする。class (s) = L_1 で class (o) = L_2 とする。このとき、$L_1 > L_2$ であるので、s は o からデータを読める。しかし、s は o に書込みを行えない。何故ならば、s が持つ最大機密 (t) の情報が、重要機密 (s) のデータベースに記憶されることにより、重要機密レベルの利用者に漏れるからである。一方、class (s) = L_2 で class (o) = L_1 ならば、s は o に書込みを行える。

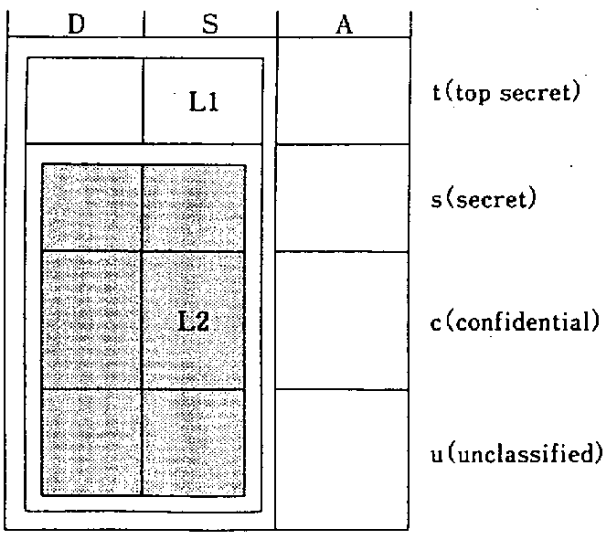


図5.5　多階層モデル

5.4　認証

分散システムでは、利用者の認証 (authentication) が重要となる。このために、暗号を

用いた方法が研究されてきている。

5.4.1 暗号

平文という利用者が理解できる文字列を他の記号列に、ある鍵を用いて変換することが暗号化である。変換された記号列を暗号文といい、暗号化のために用いられる鍵を暗号化鍵という。暗号文は、ある鍵(復号化鍵といい、暗号鍵と同一とは限らない)を用いて、あるアルゴリズムにより元の平文に戻される。これを復合化いう。復号化鍵を知らない利用者は、暗号文を元の平文に変換できない。

A．慣用暗号

送信側と受信側で共通の鍵を用いる暗号システムを、慣用暗号システムという。暗号化と復号化の手順は、公開されるが、鍵は送受信者間で秘密に管理される。米国のデータ暗号化企画である DES (Data Encryption Standard) が慣用暗号としてよく知られている。

B．公開暗号

公開鍵 (public key) 方式では、秘密鍵 E と公開鍵 D の二種の鍵により暗号化と復号化を行う。秘密鍵とは、各利用者毎に与えられ、他の利用者には秘密である。ここで、任意のメッセージ m に対して、 $m = E(D(m)) = D(E(m))$ である。一方、公開鍵は、他の利用者が知ることができる鍵である。また、暗号化と復号化のためのアルゴリズムは公開されている。

a．認証性

二人の利用者 A と B の間での通信を考える。A と B は、おのおの秘密鍵 E_A と E_B、公開鍵 D_A と D_B を持つとする。A はメッセージ m を秘密鍵 E_A で暗号化したメッセージ c ($= E_A(m)$) を B に送信する。B は、A の公開鍵 D_A により c からメッセージ m($= D_A(c)$) を得ることができる[図5.6]。A の秘密鍵 E_A を知っているのは A だけであるので、A 以外の利用者はメッセージを B に送信できない。このように、A から B にメッセージを送信するときに、A だけからメッセージが届けられるならば、通信は認証性 (authenticity) があるという。

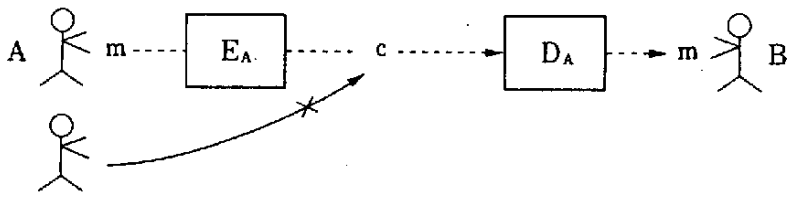


図5.6　認証性

b．秘密性

図5.7で通信の認証性が保証されていても、Aの公開鍵を知っている利用者は、Aからのメッセージcを復号化できてしまう。いわゆる、盗聴される恐れがある。このために、Aはメッセージmを Bの公開 D_B で暗号化したメッセージ $c(= D_B(m))$ を Bに送信する。Bは、自分の秘密鍵 E_B によりcからメッセージ $m(=E_B(c))$ を得ることができる[図5.7]。Bの秘密鍵 E_B を知っているのはBだけであるので、B以外の利用者はメッセージcを復号化できない。このように、AからBにメッセージを送信するときに、宛先のBだけがメッセージを受信できるならば、通信は秘密性(secrecy)があるという。

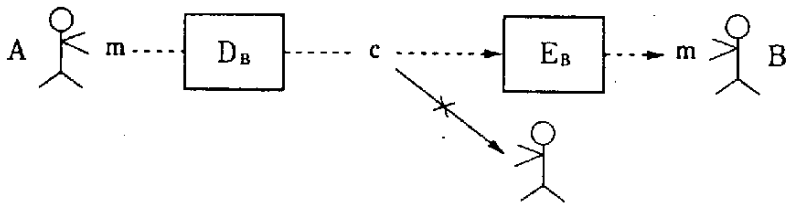


図5.7　秘密性

c．認証性と秘密性

図5.8に示すように暗号化と復号化を行うことにより、認証性と秘密性ができる。通信が認証性と秘密性を提供するとき、通信は安全であるという。

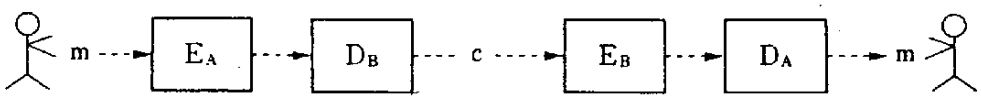


図5.8　認証性と秘密性

5.4.2 鍵の配送

分散システムで、通信を安全に行うためには、送信者と受信者間で秘密の鍵を持つ必要がある。通信しあう利用者AとBに、共通の鍵を与えることを、鍵の配送 (key distribution) という。

各利用者に鍵を配送する方式として、認証サーバ (authentication server) ASを用いる方法がある[NEED] [図5.9]。まず、各利用者は、個人鍵 (private key) を持ち、これらは認証サーバに登録されている。認証サーバASは、個人鍵を用いて利用者の認証を行う。ここで、利用者AがBと秘密な通信を行いたいとする。AとBの個人鍵をおのおのK_AとK_Bとする。

(1) 利用者Aは、認証サーバASにメッセージ<A, B>を送信する。

(2) 認証サーバASはメッセージ<A, B>を受信したならば、秘密鍵Kを生成する。ここでASは、個人鍵を用いて暗号化したメッセージ$c = K_A(K, B, T, K_B(K, A, T))$をAに送信する。ここで、Tは、ASのローカル時刻または乱数である。

(3) AがASからメッセージcを受信したならば、個人鍵K_Aを用いて復号化しKとTを得る。ついで、$K_B(K, A, T)$をBに送信する。

(4) BはAより$K_B(K, A, T)$を受信する。個人鍵K_Bを用いてKとTを得る。

(5) AとBは、秘密鍵Kを用いて通信を行う。メッセージmを送信するときに、K (m, T) として暗号化して送信する。

ここで、以前に認証サーバASから配送された鍵を用いて不正利用者が通信することを防ぐためにTが用いられる。

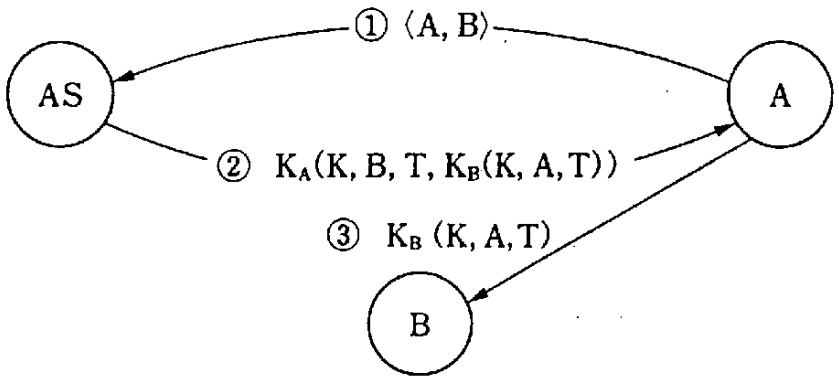


図5.9 認証サーバ

6章　まとめ

本報告では、インターネット上のオブジェクトを用いて、いかに信頼性、可用性、安全性を備えたシステムを実現していくかについて考察を行った。オペレーティングシステム(OS)と通信ネットワークが実質的に標準化された環境で、世界中に分散した情報資源をいかに、有効に利用していくかが大きな課題となってきている。インターネット上に分散したWWWサーバから情報を検索する方法は、種々研究されてきている。これに対して、インターネット上のデータベースシステム等の情報資源を用いて、いかに利用者の必要とするサービス品質(QoS：Quality of Service)を提供するシステムを構築するかがより重要な課題となってきている。本調査研究では、サービス品質として、信頼性、可用性、安全性を考え、これを提供する方法を検討した。

信頼性と可用性については、オブジェクトを多重化することにより、オブジェクトの障害に対処する方法を示した。これまで、完全多重化(能動的多重化)により、利用者に高度の信頼性と可用性を提供することは、Sybase 10のレプリケーションサーバ、ISISのIDB等で試みられている。ここでは、これらの能動的多重化に対して、すべてのオブジェクトが同一の動作を行わなくてもよい半能動的多重化方法を示した。

通信レベルの安全性は、暗号化等により達成することが種々試みられている。ここでは、オブジェクトのレベルでのアクセスを行うアクセス制御について考えた。特に、オブジェクト間の情報の流れ(フロー)を制御するためのアクセス制御として、強制アクセス制御(mandatory access control)について述べた。

以上により、インターネット上の複数のオブジェクトを用いて、利用者の要求する信頼性、可用性、安全性を提供する基本的を方式を明らかにできた。

今後の課題として、利用者の要求の変化、多様化に対して、さらにシステム環境の変化に柔軟に適応できる『やわらかい』分散システムが求められる。

参考文献

[BARB93] Barborak, M. and Malek, M., "The Consensus Problem in Fault-Tolerant Computing," ACM Computing Surveys, Vol.25, No.2, 1993, pp.171-220.

[BERN87] Bernstein, P. A., Hadzilacos, V., and Goodman, N., "Concurrency Control and Recovery in Database Systems," Addison Wesley, 1987.

[BIRM91] Birman, K., Schiper, A., and Stephenson, P., "Lightweight Causal and Stomic Group Multicasr," ACM Trans. Computer Systems, Vol.9, No.3, 1991, pp.272-314.

[BLOU89] Blough, D. M., Sullivan, G. F., Masson, G. M., "Fault Diagnosis for Sparsely Interconnected Multiprocessor Systems," Proc. 19th IEEE FTCS-19, 1989, pp.62-69.

[BUDH94] Budhiraja, N., Marzullo, K., Schneider, B. F., and Touerg, S., "The Primary-Backup Approach," Distributed Computing Systems, ACM Press, 1994, pp.199-221.

[CARE91] Carey, J. M. and Livny, M., "Conflict Detection Tradeoffs for Replicated Data," ACM TODS, Vol. 16, No.4, 1991, pp.1-12.

[CASA93] Casavant, T. L. and Singhal, M., "Distributed Computing Systems," IEEE Computer Society Press, 1993.

[CERI84] Ceri, S. and Pelagati, G., "Distributed Databases: Principles and Systems," McGraw Hill, 1984.

[CHAND83] Chandy, K. M., Misra, J., and Haas, L. M., "Distributed Deadlock Detection," ACM Trans. Computer Systems, Vol.1, 1983, pp.144-156.

[CHAND85] Chandy, K. M. and Lamport, L., "Distributed Snapshots: Determining Global States of Distributed Systems," ACM TOCS, Vol.3, No.1, 1985, pp.63-75.

[CHEN86] Chen, P.P.-S., "The Entity-Relationship Model: Toward a Unified View of Data," ACM Trans. on Database Systems, Vol.1, No.1, 1976, pp.9-36.

[CODD70] Codd, E. F., "A Relational Model for Large Data Banks," CACM, Vol.13,

No.6, 1970.

[DENN82] Denning, D. E. R. and Denning, ***., "Cryptography and Data security," Addison-Wesley, 1982.

[DU89] Du, W. and Elmagarmid, "Quasi Serializability: A Correctness Criterion for Global Concurrency Control in Heterogeneous Database Systems," Proc. of the VLDB, 1989, pp.347-355.

[ELLI91] Ellis, C. A., Gibbs, S. J., and Rein, G. L., "Groupware," Comm. ACM, Vol.34, No.1, 1992, pp.38-58.

[ELMA85] Elmasri, R., Weeldreyer, J., and Hevner, A., "The Category Concept: An Extension to Entity-Relationship Model," Data and Knowledge Engineering, North-Holland, 1985, pp.75-116.

[EPPI91] Eppinger, J. L., Mummert, L. B., and Spector, A. Z., "Camelot and Avalon." Morgan Kaufmann, 1991.

[FISC85] Fischer, M. J., Lynch, N. A., and Paterson, M. S., "Impossibility of Distributed Consensus with One Faulty Process," JACM, Vol.32, No.2, 1985, pp.374-382.

[GARC82] Garcia-Molina, H., "Elections in a Distributed Computing System," IEEE Trans. on Computers, Vol.31, No.1, 1982, pp.48-59.

[GARC85] Garcia-Molina, H. nd Barbara, D., "How to Assign Votes in a Distributed System, JACM, Vol.32, No.4, 1985, pp.841-860.

[GARC87] Garcia-Molina, H. and Salem, K., "Sagas," Proc. of the ACM SIGMOD Conf., 1987, pp.249-259.

[GRAY78] Gray, J., "Notes on Database Operating Systems," Lecture Notes in Computer Science, No.60, 1978, pp.393-481.

[GRUB94] Bruber, R., Kaashoek, F., Liskov, B., and Shira, L., "Disconnected Operations in Thor Object-Oriented Database Systems," Proc. of IEEE Workshop on Mobile Computing Systems and Applications, 1994, pp.13-24.

[HEIM85] Heimbigner, D. and McLeod, D., "A Federated Architecture for Information Management," ACM Trans. on Office Information Systems, Vol.3, No.3, 1985, pp.253-278.

[HO82] Ho, G. S. and Ramamoorthy, "Protocols for Deadlock Detection in Distributed Database Systems," IEEE Trans. Software Eng., Vol.***, No.11, 1982, pp.554-557.

[HOLT72] Holt, R. C., "Some Deadlock Properties of Computer systems," ACM Computing Surveys, Vol.4, No.****, 1972, pp.179-196.

[HUAN94] Huang, Y., Sistla, P, and Wolfson, O., "Data Replication for Mobile Computers," Prof. of ACM SIGMOD, 1994, pp.13-24.

[IEEE87] Proc. of the IEEE, Special Issue on Distributed Database System, 1987.

[JING95] Jing, J., Bukhres, O., and Elmagarmid, A., "Distributed Lock Management for Mobile Tranactions," Proc. of the ICDCS-15, 1995, pp.118-125.

[KIM85] Kim, W., Reiner, D., and Batory, D., "Query Processing in Database Systems," Springer-Verlag, 1985.

[KIM90] Kim. W., "Object-Oriented Databases: Definitions and Research Directions," IEEE Trans. of Knowledge and Data Engineering, Vol.2, No.3, 1990.

[KIST92] Kistler, J. J. and Satyanaranyanan, H., "Disconnected Operations in the Coda File System," ACM TODS, Vol. 10, No.1, 1992, pp.2-25.

[KNAP87] Knapp, E., "Deadlock Detection in Distributed Databases," ACM Computing Surveys, Vol.19, No.4, 1987, pp.303-328.

[KUNG81] Kung, H. T. and Robinson, J. T., "On Optimistic Methods for Concurrency Control," ACM TODS, Vol.6, No.2, 1981, pp.213-226.

[LAMP78] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," Comm. ACM, Vol.21, No.7, 1978, pp.558-564.

[LAMP82] Lamport, R., Shostak, R., and Pease, M., "The Byzantine Generals Problems," ACM Trans. on Programming Language and Systems, Vol.4, No.3, 1982, pp.382-401.

[LAND82] Landers, T. and Rosenberg, R., "An Overview of Multibase," Distributed Databases (Schneider, H. -J. ed.), North-Holland, 1982, pp.153-184.

[LISK88] Liskov, B., "Distributed Programming in Argus," Comm. ACM, VOl.31, No.3, 1988, pp.300-312.

[LITW86] Litwin, W. and Abdellatif, A., "Multidatabase Interoperability," IEEE Computer, No.12, 1986, pp.10-18.

[LU94] Lu, Q. and Satyanaranyanan, M., "Isolation-Only Transactions for Mobile Computing," ACM Operating Systems Review, Vol.28, No.2, 1994, pp.81-87.

[LYNC93] Lynch, N., Merritt, M., Weihl, W., and Fekete, A., "Atomic Transactions," Morgan Kaufmann, 1993.

[MAEK85] Maekawa, M., "A $\sqrt{n}$ Algorithm for Mutual Exclusion in decentralized Systems," ACM Trans. Computer Systems, Vol.3, No.2, 1985, pp.145-159.

[MAEK91] 前川、所、清水, "分散オペレーティングシステム." 共立, 1991.

[MANA93] 真鍋義文, "分散チェックポイント・ロールバックアルゴリズム," 情報処理, Vol.34, No.11, 1993, pp.1366-1374.

[MATT89] Mattern, F., "Virtual Time and Global States of Distributed Systems," in Parallel and Distributed Algorithms (Cosnard, M. and Quinton, P. eds.), North-Holland, Amsterdam, 1989, pp.215-226.

[MOWB95] Mowbray, T. J. and Zahari, R., "THe Essential CORBA," Wiley, 1995.

[MULL89] Mullender, S., "Distributed Systems," ACM Press, 1989.

[OMG93] Object Management Group (OMG), "Object Management Architecture Guide," John Wiley and Sons, 1993.

[OMG94] Object Management Group (OMG), "Common Object Service Specifications," John Wiley and Sons, 1994.

[OSZU91] Ozsu, M. T. and Valduriez, P., "Principles of Distributed Database Systems," Prentice-Hall, 1991.

[PREP67] Preparata, F., Metze, G., and Chien, R., "On the Connection Assignment Problem of Diagnosable Systems," IEEE Trans. Elect. Comput., Vol.EC-16, No.6, 1967, pp.848-854.

[RAND75] Randell, B., "System Structure for Software Fault Tolerance," IEEE Trans. Software Eng., Vol.29, No.2, 1975, pp.220-232.

[RAYN88] Raynal, M., "Distributed Algorithms and Protocols," John Wiley & Sons, 1988.

[RAYA92] Raynal, M., "About Logical Clocks for Distributed Systems," ACM Operating Systems Review, Vol.26, No.1, 1992, pp.41-48.

[RICA81] Ricart, G. and Agrawala, R. K., "An Optimal Algorithm for Mutual Exclusion in Computer Networks," Comm. ACM, Vol.24, No.1, 1981, pp.9-17.

[SCHA86] Schantz, R. E., Thomas, R. T., and Bono, G., "The Architecture of the Cronus Operating System," Proc. of the 6th Int'l Conf. on Distributed Computing System (ICDCS-6), 1986, pp.250-259.

[SCHN94] Schneider, B. F., "Replication Management Using the State-Machine Approach," Distributed Computing Systems, ACM Press, 1994, pp.169-197.

[SHET90] Sheth, A. and Larson, J.A., "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," ACM Computing Surveys, Vol.22, No.3, 1990, pp.183-235.

[SHIRA96] 白鳥則郎, 滝沢 誠, "分散処理," 丸善, 1996.

[SKEE81] Skeen, D., "Nonblocking Commit Protocols," Proc. of the ACM SIGMOD, 1981.

[SKEE83] Skeen, D. and Stonebraker, M., "A Formal Model of Crash Recovery in Distributed System," IEEE Trans. Software Eng., Vol. SE-9, No.3, 1983.

[SMIT81] Smith, J. M., Bernstein, P. A., et al., "Multibase - Integrating Heterogeneous Distributed Database Systems," Proc. of AFIPS, 1981, pp.487-499.

[SOMA89] Somani, A. K. and Agrawa, V. K., "Distributed Syndrome Decoding for Regular Interconnected Structure," Prod. of 19th IEEE FTCS-19, 1989, pp.70-77.

[TAKI80] Takizawa, M. and Hamanaka, E., "Query Translation in Distributed Databases," Proc. of the IFIP'80, 1980, pp.451-456.

[TAKI83] Takizawa, M., "Distributed Database System - JDDBS," JARECT Vol.7, Computer Science and Technologies (Kitagawa, T., ed.), Ohmsha and North-Holland, 1983, pp.262-283.

[TAKI93] 滝沢 誠, 中村章人, "放送型通信アルゴリズム," 情報処理, Vol.34, No.11, 1993, pp.1341-1349.

[TAKI96] 滝沢 誠, "リレーショナルデータベースシステム RDBMS 技術解説," ソフト・リサーチ・センタ, 1996.

[TANE92] Tanenbaum, A. S., "Modern OPerating Systems," Prentice-Hall, 1992.

[THOM79] Thomas, R. H., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases, ACM Trans. Database systems, Vol.4, No.2, 1979, pp.180-209.

[THOMA90] Thomas, L. C. and Mukesh, S., "The Delta-4 Distributed Fault-Tolerant Architecture," istributed Computing Systems, IEEE CS Press, 1990, pp.223-247.

[TSIC78] Tsichritzis, D. and Klug, A., "The ANSI/X3/SPARC DBMS Framework," Information Systems, Vol.3, No.4, 1978.

平成９年３月発行

発　行　財団法人　データベース振興センター
東京都港区浜松町２丁目４番１号
世界貿易センタービル７階
TEL　03（3459）8581

委託先　株式会社　新世代システムセンター
東京都新宿区市ヶ谷富久町
11番５号1407
TEL　03（3355）5184

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
TEL　03（3357）7161