59-S-001

高密度通信処理における分散情報統合利用システムに関する研究開発報告書

昭和60年3月

JIPDEC

財団法人　日本情報処理開発協会

この報告書は，日本自転車振興会から競輪収益の一部である機械工業振興資金の補助を受けて昭和５９年度に実施した「高密度通信処理における分散情報統合利用システムに関する研究開発」の成果をとりまとめたものであります。

序

今後の組織体等における情報処理システムは，大型コンピュータを中心とする大規模システムとは別に，パーソナル・コンピュータなどの小型コンピュータを中心とする小規模システムによって独自の処理，データの分散利用が急速に進むものと思われる。しかし，システムの有用性の観点からは，これらシステムが分散して持つ情報を相互に統合利用することが重要とされている。

このためデータ利用の基本ルーツとなるデータベースに加え，これら小規模システムを有機的に結合するローカル・ネットワーク，各種情報処理機器の持つ非コンピュータ・リーダブルなデータ（文書，書類，メッセージ等）の取扱いなど，異種のシステムに分散する情報の統合利用について3年計画で研究し，ローカル情報システムとしてモデルを開発する。

本報告書は，第2年度として，ローカル情報システム（ＬＩＳ）の基本的な通信プロトコルの実現結果，パーソナル・コンピュータ用の関係型ＤＢＭＳの実現，改良，評価の結果，異種ＤＢＳを統合し，インタオペラブルなＤＢＳとしていくための上位のプロトコルの検討結果についてまとめたものである。

本報告書がこの方面に興味をお持ちの方々に広く利用され，今後の情報処理技術向上の一助として寄与することができれば幸いである。

昭和６０年３月

目　　次

1. はじめに

最近の計算機技術の進歩により，高性能かつ高機能な超小型計算機が普及し，各作業者が，計算処理能力と，データの蓄積能力を持てる様になってきている。一方，従来からの広域のパケット変換網（例，DDX－P，VENUS－P ）の発展に加えて，Ethernet 〔METCR７６〕等のローカルエリア網（ＬＡＮ）が，重要な通信手段となってきている。ＬＡＮは，ビル，工場といったローカルなエリア内の端末，計算機，種々のデータ通信機器（例，プリンタ，センサ）を相互結合する為の一般的な手段となってきている。今後の情報システムは，ローカルなサイト毎に機器を接続し，更にこれらが広域網（ＷＡＮ）により接続された形態をとっていくものと思われる。

こうして，種々の機器が接続されることによって，共有される資源としては，データベースが中心となってきている。多くの計算機応用が，データベース中心になされてきているからである。本事業において構築を目指しているローカル情報システムは，パソコン，大型機等の種々の計算機に実現されてきたデータベースシステム（ＤＢＳ）を，まずローカルなエリアで結合し，統合的な利用を行えるようにする為のシステムである。本システムの利用者は，複数のＤＢＳを，あたかも一つのＤＢＳの様にアクセス出来る。この為に，本年度は，以下の作業を行った。

(1) パソコン用の関係型のデータベース管理システム（ＤＢＭＳ）“Granada”の開発，改良，評価。本ＤＢＭＳは，各利用者のワークステーションとしてのパソコンの作業用のＤＢＭＳとなる。

(2) ＬＡＮ上の実体（プロセス）間で，高信頼，順序保存，非同期なデータ転送を行わせるトランスポート層の通信プロトコル（トランスポート群制御プロトコル（ＴＣＣＰ））の実現

(3) (2)の上位の分散型データベースシステム（ＤＤＢＳ）プロトコルの基本設計。

(4) ＤＢＳのインタオペラビリティ（相互運用性）を実現する為のＤＢＳ統合化技術の検討。

(1)については，昨年度パソコンＰＣ－９８００で開発したものに，（group-by）集約関数（例，aver，sum，count），等の機能強化をし，PC—９８００に加えて，スーパーパソコンSun２／１２０（Unix 4.2 bad）に移植した。

(2)については，Ethernetを用いて，Ｕ－１２００（富士通）とSun２／１２０に，

Unix の利用者プログラムとして実現し，実働している。(3)については，(2)の通信プロトコルを用いて，問合せの分散処理，トランザクションの同時実行制御，信頼性制御方式を検討し，その基本設計を行い，来年度実行予定である。(4)については，ＤＢＳのインタオペラビリティの基本概念の整理を行った。特に，ＤＢＳの相互の相違として，データモデルを取り上げ，この解決方法の検討を行った。

本報告書では，まず，第２章で，ローカル情報システム（ＬＩＳ）の基本概念及び，分散型データベースシステム（ＤＤＢＳ）としてのＬＩＳの必要とする機能と，その実現の為の基本設計について述べる。第２章では，ＬＩＳのトランスポート層のプロトコルとしてのトランスポート群制御（ＴＣＣ）プロトコルの概念と実現について述べる。第３章では，パソコン用ＤＢＭＳとしてのGranadaの実現，改良，評価について述べる。

2. ローカル情報システム（ＬＩＳ）の設計思想

本章ではローカル情報システム（ＬＩＳ）を構築するための基盤となる設計思想について解説する。

2.1 分散型データベースシステム

ＬＩＳは既存のタイプの異なる（異種の）データベースシステム（ＤＢＳ）を統合化した分散型データベースシステム（ＤＤＢＳ）である〔図2.1〕。この節では，機能的にＤＤＢＳの形式的な定義を行ない，ＤＤＢＳを構築するための概念及び手法について述べる。

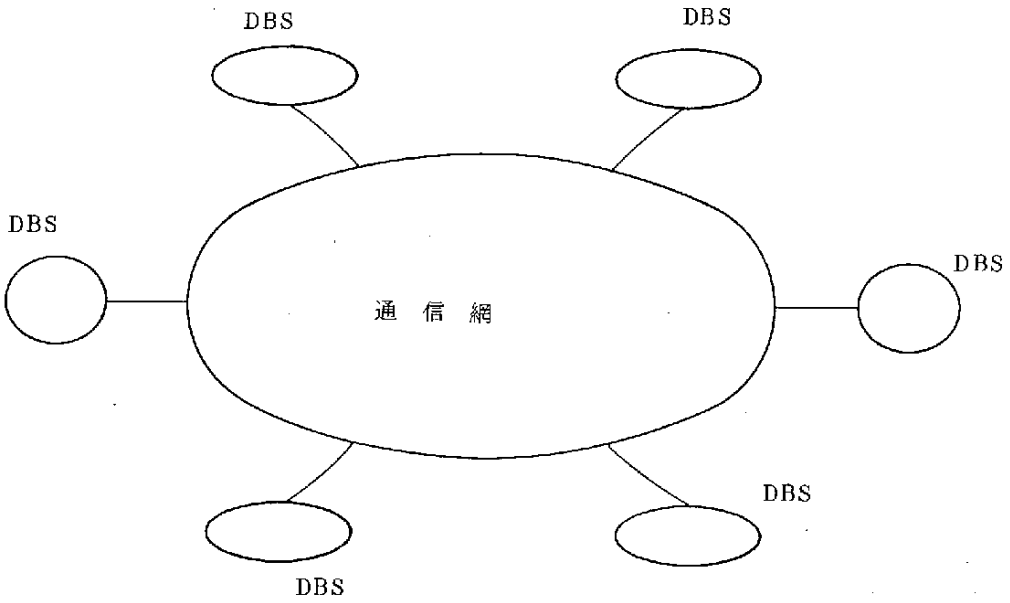


図2.1 分散型データベースシステムの構成

2.1.1 目 標

ＬＩＳは大型汎用計算機から小規模のワークステーションの既存のＤＢＳを統合化したＤＤＢＳを目指している。既存のＤＢＳはそれぞれタイプにより異なるため，ＤＢＳやＤＤＢＳの定義，また統合化の概念について定義しておく必要がある。統合化の目指す目標はいくつか考えられるが，ここでは主に次の点を目標に設定する。

(1) 既存ＤＢＳの不変性

既存ＤＢＳを変えることなしに，統合化を行なう。したがって従来のＤＢＳの操作はそのままの形でも行える。

(2) 局所性の独立

統合化されたＤＤＢＳは，各ＤＢＳの物理的な位置を意識させない。

(3) 利用形態の単一性

ＤＤＢＳとしての利用形態は単一であり，特に各ＤＢＳを意識しない操作を行ないたい場合は，単一の利用者インターフェースを提供する。

(4) 安全性（セキュリティ）の管理

ＬＩＳでは様々な情報の機密保護を行なう安全性の管理機構を有する。

(1)はＬＩＳの前提となる動機であり，何もないところからトップダウン的にＤＤＢＳを構築するのではなく，既存ＤＢＳを組み上げていく議論が必要となる。したがって全体のデータベースのトップダウン的な設計を行なわない統合化の概念が大きな問題となる。(2)は物理的には，網（ネットワーク）により，行なわれる。(3)は(2)に対して，論理的な統合化であり，情報の位置や操作性を意識させないことであり，この場合"同じ"ということの意味が大きな問題である。(4)は複合化システムには必ず起こる問題であり，ＤＤＢＳ全体で一括した管理方式が必要である。次の節では，上の目標を達成する上で必要な諸定義－データベース，データベースシステム，分散型データベースシステム，異種性（同種性），統合化などの概念について定義を行なう。

2.1.2 諸定義

データベースシステム（ＤＢＳ）の定義をここでは，そのＤＢＳが採用しているデータモデルに着目し，それにより形式化する。

データモデルとは，現実世界の限られた対象を表現したものであり，次の3つの要素から定まるものとする〔Date 81〕。

① データベース

② データベースの上での操作演算

③ インテグリティ制約

データベース（ＤＢ）とは，データの論理構造を表わすスキーマ（時間不変）（データ構造ともいう）とスキーマに実際のデータ値を与えたデータベ

ース実現値から成る。

DB：＝＜スキーマ，ＤＢ実現値＞

インテグリティ制約とはスキーマに対して，与えるデータが満足すべき条件であり，ＤＢ実現値がこれを満足していない時，矛盾がある（inconsistent）という。逆に満足している時，無矛盾である（consistent）という。

操作演算とは，検索（ＤＢ実現値よりデータの導出）及び，更新（ＤＢ実現値を無矛盾に変化させる）から成る。

現実世界を表現するには（データモデルを得るには），そのわく組となる道具が必要である。そこでデータモデルを構築するための道具をデータモデル型と定義する。データモデル型の実際の例としては，CODASYL 型，階層型，関係型などがある。

データベースシステム（ＤＢＳ）とはデータモデルを実際の計算機上に実現した処理システムと定義する。具体的にはＤＢＳは次のものから構成される。

① データベースを実現したデータベース記憶媒体

② 操作演算及び，システムの無矛盾性を保証するためのソフトウェアーデータベース管理システム（ＤＢＭＳ）

③ ＤＢＭＳを動作させるために必要なデータベース，操作演算，インテグリティ制約についての情報（ＤＤ／Ｄ）

２つのＤＢＳが異種であるというのは，その２つのＤＢＳが採用しているデータモデル型が異なっている時である。逆に同種であるというのはデータモデル型が同じである時である。

複数のＤＢＳs の統合化とは，各ＤＢＳを合せて１つのＤＢＳに見せること，すなわち，共通のデータモデル型を利用者に提供することと定義する。統合化されたものを，分散型データベースシステム（ＤＤＢＳ）とする。ＤＤＢＳにおいて，統合化されたＤＢＳs の任意のある２つが異種である時，異種のＤＤＢＳという。ＤＤＢＳが異種でない時，同種のＤＤＢＳという。

安全性制御とは，ＤＢＳ（又はＤＤＢＳ）内の情報の不正アクセスを制御するものである。制御の形態としては，アクセス権を設定して，制御を行なうものーアクセス制御，情報の不正な流れを制御するものー情報流れ制御，統計情報や論理的な推論による情報の盗用を防ぐものー推論制御，暗号化などがある。

2.1.3　同種化と統合化の手法

ここでは前節で定義した統合化の意味に即して，実際の統合化の手法について述べる。

ＤＢＳの統合化は，各ＤＢＳの各データモデルに対して，共通の型のデータモデルを利用者に提供することであり，これを行なうためには，各ＤＢＳのデータモデル型（これを局所データモデル型と呼ぶ）を共通の統一されたデータモデル型（これを全体データモデル型と呼ぶ）に変換することが考えられる。全体データモデル型を利用者に共通に与えることで統合化が達成されたとみることができる。

2.1.3.1　データモデル型の変換

データモデル型Ａをデータモデル型Ｂに変換するには，Ａのデータ構造，操作，インテグリティ制約をＢのそれで対応させればよい。この対応はそれぞれ１：ｎの形である。なぜなら，Ａにおける基本単位となる操作をＢにおいてできなければならないからである。インテグリティ制約は，更新操作においてチェックする方法があり，対応させた操作に対してそのまま適用することが考えられる。したがって本質的な問題はデータ構造及び操作の対応のさせかたである。

ここでは，データモデル型の与えるデータ構造と操作に着目して，抽象データ型としデータモデル型を代数により形式化する〔Goguen7 8〕。データモデル型間の変換は，代数間の変換となり，これはデライバの概念により形式化される。

すなわち次の２つのステップによる統合化の実現が考えられる。

① 各局所データモデル及び全体データモデルを型を通して，抽象データ型とし，代数表現する。

② 各局所データモデルの代数から全体データモデルの代数へデライバに基づき，代数変換を行なう。

2.1.3.2　データモデル型の抽象データ型化

ここではデータモデル型として，関係型とCODASYL 型を取り上げ，抽象データ型化（代数表現）することを考える。

関係型は数学的基盤をもっているため，比較的容易に代数表現すること

ができる〔Emden8 1〕。CODASYL 型の場合は順序関係や実際の記憶域に対する制御など，物理的な色彩が濃いため概念的にそのまま表現することは不可能である。そこで，ここでは，CODASYL 型を一段抽象化した，概念CODASYL 型〔Takizawa 8 3 a，8 4 d〕を代数化することを考える。

抽象データ型の代数による表現は形式化，検証などにおいて，特にソフトウェア工学などで，その正当性が評価されている〔Goguen 7 8〕。通常，代数は台（carrier）と呼ばれる集合と，その上で働く，いくつかの演算（operation）とで定義される。ところが現実には，台は複数の種類を必要とする場合が多い。そこで複数の種類の台を有する代数が考えられる。つまり台の種類を表わすソートという概念を持ち込み，ソートの数が複数ある代数は多ソート代数と呼ばれている。形式的には，まず多ソート代数を与えるものとしてシグニチャが定義される。すなわち，台の種類を表わすソート s の集合 S と，S の上で定義される演算記号集合の族 Σ から構成される＜S，Σ＞をシグニチャと呼ぶ。これに対して，各ソート s に台 As（ソート s の台を As と書く），各演算記号にその上で定義される関数（定数を含む）を割り当てることにより，多ソート代数が定義される。さらに，定義された関数の機能を表わすのに，別の関数との関係を等式の集合とで表現する。ε を公理と呼び，シグニチャ＜S，Σ＞を合せた＜S，Σ，ε＞を，代数を表わす仕様と呼ばれる。

〔例〕

ソート集合として S＝｛ bool，real，int ｝

演算記号集合の族 Σ として

Σ＝｛ Σ bool real real，real，Σ int real，real ｝

Σ bool real real，real ＝｛ COND ｝

Σ int real，real ＝｛ EXP ｝

COND ： bool，real，real → real

EXP ： int，real → real

各演算を図表現すると，図 2.2 のようになる。

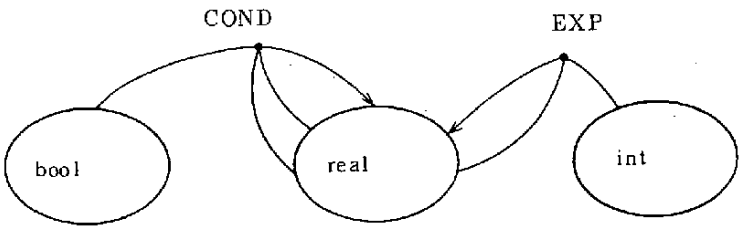


図 2.2　多ソート代数のソートと関数の関係

図による表現では通例として，各ソートには下線を引く。また，図表現した場合，矢印のもとにあるソートが関数（演算）の引数，矢印の先にあるものが値となるソートを表わす。

データモデル型を代数で仕様記述することを考えた場合，データ構造，操作演算，インテグリティ条件の3つの要素を考えなければならない。このうち，インテグリティ制約は操作の用いられかたの制約であり，これは代数で言いかえれば関数の用いられかたの制約となる。データモデルのデータ構造は単純代数の台として定義するのではなく，関数と一体化して定義される。またデータモデル上の複数の操作を代数のある関数に対応させない。これはデータモデル上の基本要素となる操作を維持させるためである。逆にデータモデル上の操作を代数上の複数の関数で対応させることはある。一般にデータモデル上の表現力は代数のそれよりも大きいからである。仕様の都合上，データモデルを複数の代数に分けて定義することがある。分けた代数は後でまとめて1つの代数にしても，一般的な代数の性質を壊わさないことが保証されている〔Goguen 7 8〕。データモデル型の代数による仕様は次のように書く。

type　抽象データ型名
　sorts　ソート1，ソート2……（ソートの定義）
　operations
　　演算名1：ソートk_1，ソートk_2，…→ソートk
　　演算名2：ソートl_1，ソートl_2，…→ソートl
　　⋮
　axioms（演算の意味を表わす等式）
　　等式1

等式2

⋮

2.1.3.3　関係型の抽象データタイプ化

関係型の代数による抽象データタイプ化は前述したように〔Emden81〕のものがある。ここでは，この仕様を基盤とし，いくつかの用語を改め，いくつか関数を省いた形で定義する。また公理は，ホーン節によるものを等式にした形で記述した。この仕様の特徴は，データベースの状態を表わすデータベース実現値（database instance）を1つのソートとして陽に定義し，現在のデータベースの状態を把握できるようにしている。また，データベースを形成する関係と，関数（関係代数）により作られる関係を区別し，更新操作においてデータベース実現値を引数としてとることにより，データベースの状態が容易には変わらないように工夫されている。よく混同される関係それ自身と関係名の区別も，データベース実現値をとる関数ととらない関数を分けることにより，行なわれる。

データモデル型の代数化にあたっては，よく使われる概念として集合がある。そこで，特別に先に定義しておくと都合がよい。関係型では，集合の概念をもつ抽象データ型として，タプルや属性などが考えられる。いわゆる集合は，共通の性質を持つ抽象データ型に対して適用され，別の抽象データ型が導出されるパラメタ型（parameterized type）と呼ばれるものである。したがって集合の性質をもつ抽象データ型であれば，これを適用して新たな集合としての抽象データ型を定義することが可能である。これは概念CODASYL型の代数化にも用いられる。

以下に関係型の代数としての仕様をあげる。また，仕様を図2.3に示した。ただし，便宜上，関係型の主要なデータ構造（属性，組，etc）ごとに代数が分けて定義されている。又，先に集合の定義がある。これにより，抽象データ型tに対する集合のソートをset of〔t〕と書くことにする。

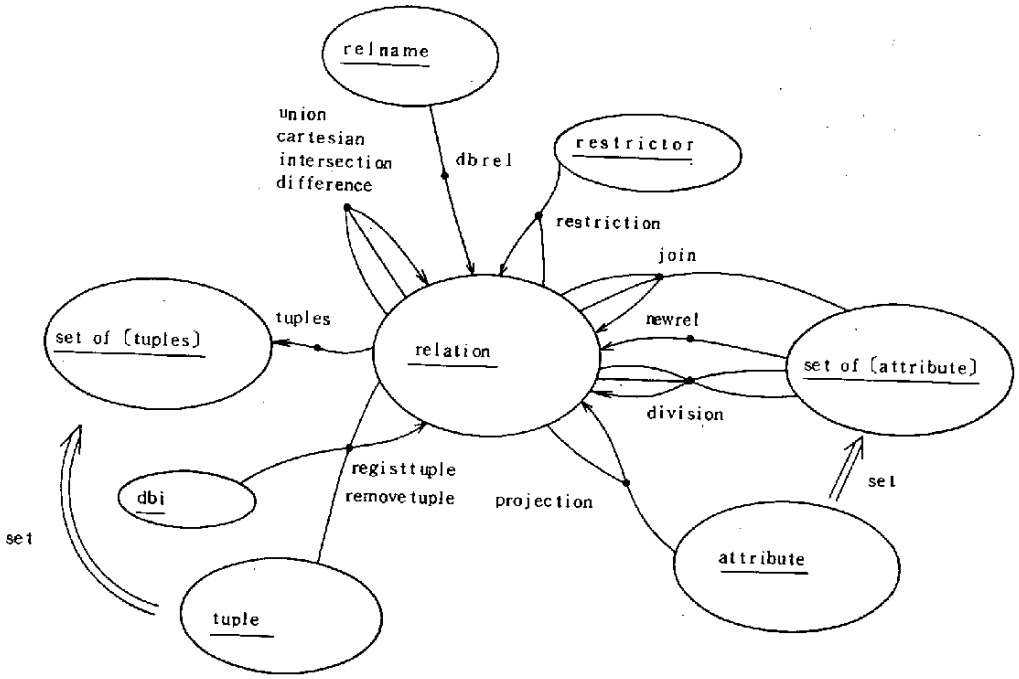


図 2.3　関係型のデータベースの代数表現

集合の定義

type set

sorts set, element, bool

operations

ϕ	nil	: → set	（空集合）
・	insert	: element, set → set	(insert)
$\in$	member	: element, set → bool	(membership)
$\subset$	subset	: set, set → bool	(inclusion)
eguiv	eguiv	: set, set → bool	(eguivalence)
$\cup$	union	: set, set → set	(union)
−	differ	: set, set → set	(difference)
$\cap$	inter	: set, set → set	(intersection)
true	true	: → bool	（真）
false	false	: → bool	（偽）

（infix記号）（演算記号）

axioms u : element s,t : set

u・s ∪ u・t ≡ u・(s ∪ t)

u・s ∩ u・t ≡ u・(s ∩ t)

u・s − u・t ≡ u − s

s ∩ t ≡ s −(s ∪ t − t)

u ∈ φ ≡ false

u ∈ u・s ≡ true

etc.

⋮

属性の定義

type attribute

sorts attribute, bool

operations

属性名１： → attribute

属性名２： → attribute

⋮

属性の定義

≈ : attribute, attribute → bool (属性の一致)

axioms a : attribute

a ≈ a ≡ true

タプルの定義

type tuple

sorts tuple, attribute, set of〔attribute〕, set of〔tuple〕

value a (属性 a に対する値), velue

operations

new : set of〔attribute〕→ tuple (空のタプルを作る)

store : tuple, attribute, value → tuple (タプルに値を設定する)

columns : tuple → set of〔attribute〕(タプルより属性集合を得る)

read : tuple, attribute → value (タプルより値を得る)

piece : tuple, set of〔attribute〕→ tuple (タプルの細分化)

catenete : tuple, tuple → tuple (タプルの結合)

compose : set of〔tuple〕, tuple→set of〔tuple〕(catenateの一般化)

match : tuple, tuple, set of〔attribute〕→bool (タプル値の一致)

eq : value, value→bool (値の一致)

in_a : $value_a$ → value (値を作る)

att : value → attribute (与えられた値に対する属性)

vel : value → $value_a$ (特定の属性に対する値)

axioms

t, t′ : tuple a, a′ : attribute A, A′ : set of〔attribute〕

s, s′ : set of〔tuple〕 v, v′ : value

columns (new (A)) ≡ A

columns (store (t, a, v)) ≡ column (t)

store (store (t, a, v), a′, v′) ≡ store (store (t, a′, v′), a, v)

$$\text{read}(\text{store}(t, a', v), a) \equiv \begin{cases} v & (a \equiv a' \text{ の時}) \\ \text{read}(t, a) & (a \not\equiv a' \text{ の時}) \end{cases}$$

piece (new(A), A′) ≡ new(A′) (A′⊃A の時)

$$\text{piece}(\text{store}(t, a, v), A) \equiv \begin{cases} \text{store}(\text{piece}(t, A), a, v) & (a \in A) \\ \text{piece}(t, A) & (a \in A) \end{cases}$$

catenate (new(A), new(A′)) ≡ new (A+A′) (A∩A′ = φ)

catenate (new(A), store (t, a, v)) ≡ store(catenate(new(A), t), a, v)

compose (nil, t) ≡ nil

compose (t・s, t′) ≡ catenate (t, t′)・compose (s, t)

match (t, t′, φ) ≡ true

match (t, t′, a・A) ≡ match (t, t′, A)

(a・A⊂ columns(t) かつ a・A⊂ columns(t′) かつ read (t, a) eq read (t′, a))

関係の定義

type relation

sorts relation, relname, dbi

operations

関係名１： → relname

関係名２： → relname

⋮

関係名ｎ： → relname

関係名１：dbi → relation

関係名２：dbi → relation

⋮

関係名ｎ：dbi → relation

データベースの定義

type database

sorts relation, relname, tuple, set of〔tuple〕, attribute,
set of〔attribute〕, restictor, dbi

operations

dbrel：relname → relation（データベースを形成する関係）

newrel：set of〔attribute〕→ relation（空の関係を作る）

registtuple：tuple, relation, dbi → relation（タプルの追加）

removetuple：tuple, relation, dbi → relation（タプルの削除）

tuples：relation → set of〔tuple〕（関係中のタプル）

union：relation, relation → relation（和）

cartesian：relation, relation → relation（直積）

intersection：relation, relation → relation（積）

difference：relation, relation → relation（差）

projection：relation, set of〔attribute〕→ relation（射影）

join：relation, set of〔attribute〕, relation → relation
（自然結合）

restriction：relation, restictor → relation（制的）

division : relation, set of〔attribute〕,
set of〔attribute〕, relation→relation
（商）

attribute : relation→set of〔attribute〕（関係を構成する属性集合）

belong : tuple→relation （タプルの属する関係）

axioms t : tuple r, s : relation A : set of〔attribute〕

A ≡ attributes (newrel(A))

tuples (newrel(A))≡ φ

tuples (registtuple (t, newrel(A))) ≡ φ・t

tuples (removetuple (t, registtuple (t, newrel (A)))) ≡ φ

tuples (union(r, s))≡tuples (u) U tuples (s)

attributes (union (r, s))≡attributes (r)≡attributes(s)

attributes (cartesian (r, s))≡attributes (r)Uattributes(s)

attributes (intersection(r, s))≡attributes(r)≡attributes (s)

tuples (intersection(r, s))≡tuples (r)∩tuples(s)

attributes (difference (r, s))≡attributes (r)≡attributes (s)

tuples (difference (r, s))≡tuples(r)－tuples(s)

attributes (projection(r, A))≡A

tuples (projection (r, A))≡(piece (t, A)・φ)U
tuples (projection (removetuple (r, t), A))

2.1.3.4 概念CODASYL 型の抽象データタイプ化

概念CODASYL 型はスキーマとして，レコード型（R）とセット型（S）を持つ。Rは項目集合Ω_R＝｛@R，t_1，…，t_m｝（m≧o）とインテグリティ制約Γ_Rとから成る（R＝（Ω_R，Γ_R））。各項目$\tau\in\Omega_R$のとり得る値の集合（定義域）をdom（τ）とする。@Rはデータベースキー（dbキー）で，dom（@R）はRの識別子集合となる。各t_iはデータ項目である。Γ_RはΩ_Rのとり得る値（レコード実現値又はレコード）を定める述語で，この集合をレコード実現値（RO）集合R＝｛o|o∈

$\mathrm{dom}(\Omega_R) \wedge \Gamma_R(r)\}$（ここで $\mathrm{dom}(\Omega_R) = \overset{m}{\underset{i=1}{X}} \mathrm{dom}(t_i) \times \mathrm{dom}(@R)$ とする。）各レコード実現値 $o \in R$ の意味のある部分項目集合 I（$\subseteq \Omega_R$）の値を $I(o)$ とする。各 $o \in R$ のdbキー $@R$ の値はRのみならず他のRo集合内でも一意であるので，$@R$ はRの主キーと考えられ，$@R(o)$ を $\delta(o)$ と表わす。各Rについて，次のR述語 $\rho_R : \mathrm{dom}(\Omega_R) \rightarrow \{\mathrm{true}, \mathrm{false}\}$ を定める。

$$\rho_R(o) = \mathrm{true}\ \underline{\mathrm{if}}\ o \in R,\ \mathrm{false}\ \underline{\mathrm{othewise}}$$

次に，2つのレコード型間の関係としてセット型Sが定義される。すなわち，部分関数 $f_S : \mathrm{dom}(\Omega_{R_2}) \rightarrow \mathrm{dom}(\Omega_{R_1})$ が定義されているとき，$\underline{S} = (\underline{R_1}, \underline{R_2}, \Gamma_S)$ と表わす。R_1 と R_2 はそれぞれ f_S の値域と定義域を与えるレコード型であり，親，子レコード型という。Γ_S は Ω_{R_1} と Ω_{R_2} の関係より定まる述語で，これにより $\underline{S}$ に実際に与えられるレコード実現値対（セット実現値）が定まる。そしてこの集合をセット実現値（SO）集合 $S = \{u \mid u \in \overset{2}{\underset{i=1}{X}} \mathrm{dom}(\Omega_{R_i}) \wedge \Gamma_S(u)\}$ とする。各 $u = (o_1, o_2) \in S$ で，$o_1 \in R_1$，$o_2 \in R_2$ でなければならない。この時，親と子レコード実現値といい，o_1 と o_2 は親子関係にあるという。Sは f_S により与えられるので，各 $o_1 \in R_1$ は h（$\geq o$）個の子 $o_2 \in R_2$ を，また各 $o_2 \in R_2$ はたかだか1つの親 $o_1 \in R_1$ をもてる。$@R_i$ は R_i の主キーであり，識別子であるので，各 (o_1, o_2) Sをdbキーの対 $(\delta(o_1), \delta(o_2))$ によって表すこともある。以下，$S = \{u \mid u = (\delta(o_1), \delta(o_2)) \wedge u' = (o_1, o_2) \in \overset{2}{\underset{i=1}{X}} \mathrm{dom}(\Omega_{R_i}) \wedge \Gamma_S(u')\}$ とする。Sは次のS述語 $\sigma_S : \overset{2}{\underset{i=1}{X}} \mathrm{dom}(\Omega_{R_i}) \rightarrow \{\mathrm{true}, \mathrm{false}\}$ と同じ意味をもつ。

$$\sigma_S(o_1, o_2) = \begin{cases} \mathrm{true}\ \underline{\mathrm{if}}\ (\delta(o_1), \delta(o_2)) \in S \\ \mathrm{false}\ \underline{\mathrm{otherwise}} \end{cases}$$

このように，概念CODASYLモデルでは，通常のCODASYL型の親子関係の順序及び物理的な構造が省かれていて，通常のCODASYLモデルの論理的な構造が与えられている。概念CODASYL型の代数での表現は，関係型では必要のなかったセット型の定義が必要である。定義体は次のようになる。集合の概念はCODASYL型でも使用する。

以下にCODASYL型の仕様を示す。又，この仕様を図2.4に示す。

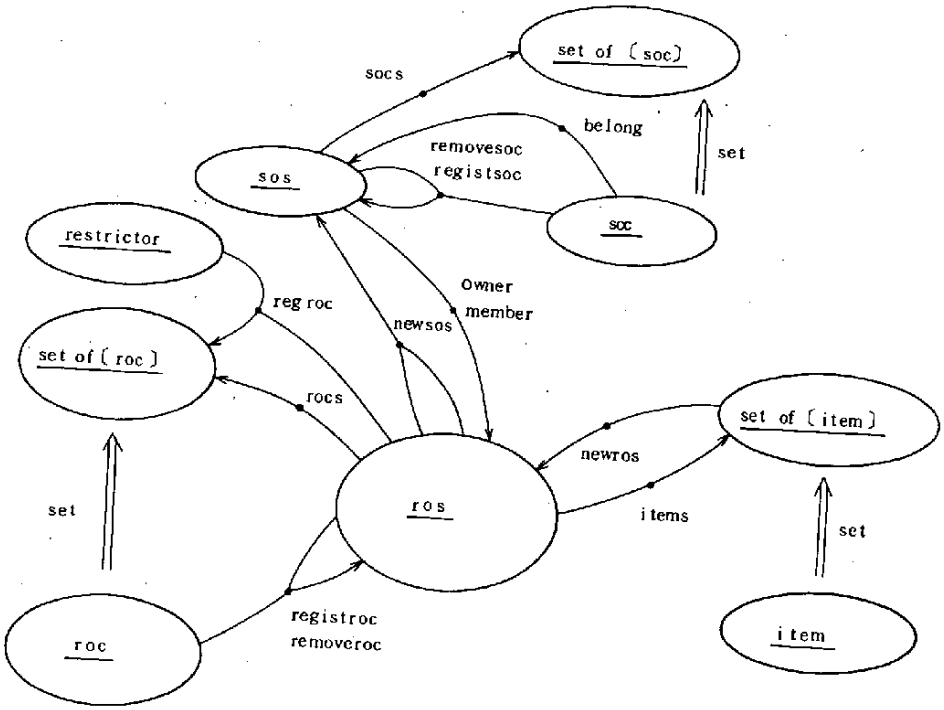


図 2.4　CODASYL型の代数表現

データ項目の定義

type　item

sorts　item, bool

operations

項目名 1.：　→ item

項目名 2：　→ item

⋮

≈　　：item, item → bool

axions　i : item

$i \approx i = \text{true}$

レコード実現値の定義

type roc　(record occurrence)

sorts roc, item, set of〔item〕, set of〔roc〕, value, bool

operations

newroc : set of〔item〕→ roc　　（空のレコードを作る）

writefield : roc, item, value → roc（データ項目に値を設定）

fields : roc → set of〔item〕　　（各データ項目を知る）

readfield : roc, item → value　　（データ項目の値を知る）

eq　: value, value → bool　　（値の一致）

axioms I : set of〔item〕 r : roc i, i' : item v, v' : value

fields (newroc (I)) ≡ I

readfield (writefield (r, i, v), i') ≡
- v　($i \approx i'$)
- readfield (r, i')　($i \gtrapprox i'$)

writefield (writefield (r, i, v), i', v') ≡
- writefield (r, i, v') ($i \approx i'$)
- writefield (writefield (r, i', v'), i, v)　($i \gtrapprox i'$)

セット実現値の定義

type soc

sorts roc, ros, set of〔roc〕, set of〔ros〕, soc, set of〔soc〕

operations

newsoc : ros, ros → soc　　（空の soc を作る）

regowner : soc, roc → soc　　（ soc で親となる roc を登録 ）

regmember : soc, roc → soc　　（ soc で子となる roc を登録 ）

ownerroc : soc → roc　　（ soc の親である roc を得る ）

memberroc : soc → roc　　（ soc の子である roc を得る ）

ownersocs : roc → set of〔soc〕　（ roc を親とする soc の集合を得る ）

membersocs : roc → set of〔soc〕（ roc を子とする soc の集合を得る ）

ownerros : soc → ros　　（ soc の親である ros を得る ）

memberros : soc → ros　　（ soc の子である ros を得る ）

axioms u_1, u_2 : ros s : soc r : roc

ownerros (newsoc (u_1, u_2)) ≡ u_1

memberros (newsoc (u_1, u_2)) ≡ u_2

ownerroc (regowner (s, r)) ≡ r

memberroc (regmember (s, r)) ≡ r

データベースの定義

type database

sorts ros, roc, set of〔roc〕, sos, soc, set of〔sos〕,
set of〔soc〕, item, set of〔item〕, value, restrictor

operations

newros : set of〔item〕→ros　（空のレコード型をつくる）

registroc : roc, ros → ros　（rocを登録）

removeroc : roc, ros → ros　（rocを削除）

rocs : ros → set of〔roc〕　（rosに属するrocの集合を得る）

reqroc : ros, restrictor → set of〔roc〕　（条件にあったrocの集合を得る）

items : ros → set of〔item〕　（rosのitem構成を得る）

newsos : ros, ros → sos　（空のセット型をつくる）

registsoc : sos, soc → sos　（socを登録）

removesoc : sos, soc → sos　（socを削除）

socs : sos → set of〔soc〕　（sosに属するsocの集合を得る）

owner : sos → ros　（sosの親であるrosを得る）

member : sos → ros　（sosの子であるrosを得る）

belong : soc → sos　（sosが属するsosを得る）

axioms I : set of〔item〕 r : roc u_1, u_2, u_3 : ros t : soc s : sos

I ≡ item (newros (I))

rocs (newros (I)) ≡ ϕ

rocs (registroc (u, r)) ≡ rocs (u) ∪ r・ϕ

rocs (removeroc (u, r)) ≡ rocs (u) − r・ϕ

removeroc (registroc (u, r),r) ≡ u

registroc (removeroc (u, r),. r)≡u

owner (newsos (u_1, u_2))≡u_1

member (newsos (u_1, u_2))≡u_2

socs (newsos (u_1, u_2))≡ϕ

socs (registsoc (s, t))≡socs (s)∪t・ϕ

socs (removesoc (s, t))≡socs (s)−t・ϕ

registsoc (removesoc (s, t), t)≡s

removesoc (registsoc(s, t), t)≡s

2.1.3.5　全体データモデル型

ここでは関係型を拡張したデータモデル型として全体データモデル型を提案する。これは次節で，関係型，CODASYL 型を統合するモデル型として用いられる。

各データモデル型はそれぞれ，表現能力が異なるため，それを統一化する全体データモデル型の表現能力は高いものが要求される。また，論理的（数学的）基盤をもつことも条件の１つである。そこで，論理的扱いが容易な関係型に，情報の記述能力の弱さを改善するため，幾つかの操作を追加した形とした。特にデータベース自身の記述（ＤＤ／Ｄデータディクショナリ・ディレクトリ）として，値，属性，組，関係を相互に変換することができると都合がよい。全体データモデル型では，これらが演算として追加されている。したがって仕様は，前述した関係型の仕様に，次に示す演算が加わることになる。ソートは基本的には変わらない。これは現存する代数に対して，新しい関数を単に強化（enrich）したものである〔Gognen 78〕。つまり，関係型の代数の仕様を＜S，Σ，E＞とすると，追加される関数に対する演算記号領域Σ'とその意味を定義する等式集合E'を加えた＜S，$\Sigma \cup \Sigma'$，$E \cup E'$＞を仕様として用いる。

以下に強化される関数と公理を示す。

operations

relattr：relation→attribute　　（関係を属性にする）

tupleval：tuple→velue　　（組を値にする）

reltuple：relation→tuple　　（関係を組にする）

valattr：value→attribute　　（値を属性にする）
tuplerel：tuple→relation　　（組を関係にする）
attrval：attribute→value　　（属性を値にする）
attrrel：attribute→relation　　（属性を関係にする）
valtuple：value→tuple　　（値を組にする）

axioms

relattr（attrrel（a））≡a　　a：attribute
tupleval（valtuple（v））≡v　　v：value
：　可能な組み合せ（逆関数により元に戻る）

2.1.3.6　全体データモデル型による統合化

ローカルなデータモデル型から全体データモデル型への変換は，代数におけるデライバの概念により行なわれる。すなわち，ある代数から別の代数への変換は，各代数上のソート間での写像fと関数間での写像dを用いたデライバ$\delta = <f, d>$で形式化される。

(1)　関係型から全体データモデル型への変換

全体データモデル型は関係型を強化したものであるから，デライバとしては恒等写像（identity mapping）を考えればよい。

(2)　概念CODASYL型から全体データモデル型への変換

概念CODASYL型の親子関係（セット）を除けば，各ソートを次のように対応させることができ，関数も対応したものが存在する。

概念CODASYL		関係型
項目（item）	⟶	属性（attribute）
レコード実現値（roc）	⟶	組（tuple）
レコード型（ros）	⟶	関係（relation）

従がって，親子関係のみに注目して考えればよい。親子関係実現値（セット実現値）はレコード間の関係であり，全体データモデル型でこれを表現するのに，各親，子レコードに対応する組を値とみて，その対となるタプルを構成する。親子関係型（セット型）も，全体データモデル型では，親，子という属性をもつ2項組を考え，レコード型に対応する関係を値に

したものをその属性値として設定することにより実現する。

以下に概念CODASYL 型から全体データモデル型へのデライバ $\delta = <f, d>$ を記述する。ただし，親子関係のみ（セット型とセット実現値）に関するものである。

概念CODASYL→TDMデライバ

ソート

f(ros)＝relation　　f(set of〔soc〕)＝set of〔tuple〕

f(roc)＝tuple　　f(set of〔sos〕)＝set of〔relation〕

f(sos)＝relation

f(soc)＝tuple

関数

- d(newsoc(r_1, r_2))＝newtuple(relattr(r_1')・relattr(r_2')・ϕ)

  r_1, r_2 : ros | r_1', r_2' : r_1, r_2 に対応する relation

- d(regowner(u, t))＝writecolumn(u', valattr(readcolumn(reltuple(belong(u'))(member), owner)), tupleval(t'))

  u : soc　t : roc | u' : uに対応する tuple　t' : tに対応する tuple

- d(ownerroc(u))＝valtuple(readcolumn(u', valattr(readcolumn(reltuple(belong(u')), owner))(member)))

  u : soc | u' : uに対応する tuple

- d(ownerros(u))＝attrrel(valattr(readcolumn(reltuple(belong(u')), owner)))(member)

  u : soc | u' : uに対応する tuple

- d(newsos(r_1, r_2))＝tuplerel(writecolumn(writecolumn(newtuple(owner・member・ϕ), owner, attrval(relattr(r_1'))), member, attrval(relattr(r_2'))))

  r_1, r_2 : ros | r_1', r_2' : r_1, r_2 に対応する relation

- d(owner(s))＝attrrel(valattr(readcolumn(reltuple(s'), owner))(member))

  s : sos(member) | s' : sに対応する relation

2.1.4　安全性（セキュリティ）

安全性制御はＤＢＳ（又はＤＤＢＳ）内の隠蔽が必要な情報を各利用者に自由な参照，更新を許さないための制御である。

安全性制御は，大別すると４つのクラス，アクセス制御，情報流れ制御，推論制御，暗号化がある。既存ＤＢＳを統合化したＤＤＢＳにおける安全性制御の統一化（各ＤＢＳで採用している安全性制御をまとめて，統一化した機構を考えること）は次の問題がある。

①　各ＤＢＳにおける安全性の対象（オブジェクトという）が異なる。
（属性値，組，関係等）

②　①の対象の相違などから，各ＤＢＳの制御機構が異なる。

③　各ＤＢＳに対して既存に設定された安全性を生かした形で，統一化した機構が必要（既存のＤＢＳをローカルに利用することがある）。

前節で述べた統合化により，データモデルの共通化が行なわれるため，データモデルを基盤とした抽象化した安全性制御の意味を考えることができる。

2.1.4.1　代数と安全性制御

データモデルは代数によって表現される。したがって安全性の対象（以降オブジェクトという）となるタイプは台（集合）であり，オブジェクトはその台の要素となる。例えば，組をオブジェクトとすれば，全体データモデルにおいて，オブジェクトとなるタイプはA tuple（tupleの台）であり，オブジェクトはA tuple中の要素となる。

オブジェクトは，各演算（検索，更新等）により操作されるが，全体データベースモデルにおいては，関数の形となる。例えば，ある関係に組を追加する場合，

$$\mathrm{registtuple}：A_{\mathrm{tuple}} \times A_{\mathrm{relation}} \rightarrow A_{\mathrm{relation}}$$

（registtuple：tuple，relation→relation）

（A_s はソート s の台）

の形となる。

アクセス制御と情報流れ制御は利用者に対して上のような関数の評価の可否を制御することで実現できる。

アクセス制御の場合，利用者とオブジェクトの間にアクセス権を設定し，当該関数の意味（動作のタイプ）とその関数に使用されている台の要素に

より，その関数を実行できるか判定し，制御する。

情報流れ制御は，あらかじめオブジェクトに安全性クラスを設定し，関数の性質－関数の左側から右側へ情報が流出する－により，各オブジェクトのクラスのチェックで行なえる。

ここでは，代数による情報流れ制御の手法を与える。

2.1.4.2　代数による情報流れ制御

不正な情報の流れは，ある利用者が，その利用者に許されている機密の情報を，他の許されていない利用者に渡すことにより起こる。したがって，オブジェクトに安全性クラスを設定し，その間での流出関係をチェックしていく方法がある。

〔Denning 76，82〕では，安全性クラスとフロー関係，基本的な演算（least upper bound ⊕と greatest lower bound ⊗）により束（lattice）でモデル化を行なっている。ここでもそれに従い，議論を進める。

今，代数の台をD_1，D_2 …，Dnとする。オブジェクトとしてはD_1，D_2 …，Dn のデータ要素$d_1 \in D_1$，$d_2 \in D_2$ … $d_n \in Dn$ とし，安全性クラスとしてそれぞれsc_1，sc_2，… sc_n を持つことにする。今，評価したい代数の関数を

$$D_1 \times D_2 \times \cdots \times D_{m-1} \rightarrow D_m$$

とすると，これは左辺の各台に含まれるデータ要素から右辺の台のデータ要素へ情報が流れると考えることができる。したがって不正な情報の流れは安全性クラス間の流れ関係をチェックすることにより調べられる。Di のデータ要素の安全性クラスをsc_iとし，Di のデータ要素をDj のデータ要素に情報を流せることを$sc_i \rightarrow sc_j$と書くことにすると，情報流れモデルとして次のものを設定する。

FM＝＜SC，→，⊕，⊗＞

SC：安全性クラスの集合

→：情報の流れ関係

A，B：安全性クラス∈SC とすると

A→B　とはクラスAの情報をクラスBに流せる。半順序集合。

⊕：least upper bound 演算子

すべての安全性クラスA，B　∈SCに対して，次の条件を満た

す安全性クラスCがただ1つあり，C＝A⊕Bと書く。

(i) A→CかつB→C

(ii) A→DかつB→Dなる安全性クラス内の全Dに対してC→D

⊗ ：greatest lower bound 演算子

すべての安全性クラスA，B∈SCに対して，次の条件を満たす安全性クラスCがただ1つあり，C＝A⊗Bと書く。

(i) C→AかつC→B

(ii) D→AかつD→Bなる安全性クラス内の全Dに対してD→C

情報流れモデルFMは束（ lattice ）を形成する。そこで，代数の関数を3つのタイプに分けた時，それぞれ次の条件を満たす時，不正な情報の流出がないと言える。

① 更新系の演算では，

$$sc_1 \oplus \cdots \oplus sc_{m-1} \rightarrow sc_m \qquad (1)$$

でなければならない。

② 検索系の演算では，関数の右側は新たに作られるDmのデータ要素であり，これを用いて，さらに不正なことが起らないよう(1)式になるようにscmを設定する。

（通常は$sc_m = sc_1 \oplus \cdots \oplus sc_{m-1}$

③ 関数の左側が存在しない，新たなデータ要素を作る場合には，その利用者に与えられているデータ要素のすべて（D_1，…，D_kの要素）に書き込むことができなければならないので，

$$sc_m \rightarrow sc_1 \otimes \cdots \otimes sc_k \qquad (2)$$

となるようにD_mのデータ要素の安全性クラスsc_mを設定しなければならない。

（通常は$sc_m = sc_1 \otimes \cdots \otimes sc_k$）

例．データベース中の関係をA，B，Cとし，各安全性クラスを

$sc_A \rightarrow sc_B$　　$sc_B \rightarrow sc_C$とする。

$sc_A \rightarrow sc_B \rightarrow sc_C$

(i) Cを検索して，その結果（関係）とBの和（ union ）をとり，それをBとする。

・ Cの検索

restriction：C×制約子→C′（検索結果）

上の②より，C′ の安全性クラス $sc_{C'}$ を

$sc_{C'} = sc_C$　（制約子はオブジェクトと考えない）

・ C′ とBの和をB

union：C′×B　→　B

①より $sc_{C'} \oplus sc_B \rightarrow sc_B$ でなければならない。

$sc_{C'} \oplus sc_B = sc_C \oplus sc_B = sc_C$

ところが $sc_C \rightarrow sc_B$ でないのでこの操作は許されない。

(ii) Aを検索して，その結果（関係）とCの和（union）をとり，それをCとする。

・ Aの検索

restriction：A×制約子→A′（検索結果）

②よりA′ の安全性クラス $sc_{A'}$ は $sc_{A'} = sc_A$

・ A′ とCの和をC

union：A′×C　→　C

①より $sc_{A'} \oplus sc_C = sc_C$ でなければならない。

$sc_{A'} \oplus sc_C = sc_A \oplus sc_C = sc_C$

$sc_C \rightarrow sc_C$ より，この操作は許される。

2.2　分散型データベースシステムとトランザクション管理

この節では分散型データベースシステム（ＤＤＢＳ）におけるトランザクション管理の方法を同時実行制御，信頼性方式を中心として述べる。

2.2.1　諸概念とモデル化

データベースシステム（ＤＢＳ）の特徴として，共有性と統合性があるが，これに基づき多数の利用者が同時に同じデータベースをアクセスできることが必要である。つまり，ＤＢＳの性能（スループット）を高めるために複数利用者による同時アクセス，即ち演算をインターリーブして実行する。その時，問題となるのは同時アクセスによる矛盾のあるデータの生成であり，秩序だったアクセスの制御が必要となる。性能向上のための同時アクセスを許し，かつ，システム全体を無矛盾な形に保つための制御を同時アクセス制御

（一般的には同時実行制御）という。

ＤＢＳは同時に，システムの障害に対する予防や，実際に障害が起こった時に，使用できる資源の最大利用，回復する手段をもつことが必要である。システムの信頼性とはこれらの問題に対するシステムの対処の度合いを示すものであり，信頼性制御とは，システムの信頼性を高めるための制御である。

同時実行制御，信頼性制御の機構を考える上で，利用者の要求（問い合せ，更新）を１つのまとまった形ートランザクションーとしてとらえて，これを管理する手法が一般的である。トランザクションをもとにして各ＤＢ，DBS，ＤＤＢＳ等を抽象化してとらえたものをトランザクション処理モデルという〔Bernstein８１〕。分散型データベースシステム（ＤＤＢＳ）はこの場合，網（ネットワーク）でつながったサイトの集合体となる。サイトとは大きく分けると３つの構成要素からなる。

(1) トランザクション管理（ＴＭ）

利用者の要求をトランザクションとして管理し，スケジューラ間とやりとり，又は他のサイトとやりとりするものである。これはさらに細かい部分で構成される（2.2.3項参照）。

(2) スケジューラ

ＴＭから送られた命令の順番を調整し，データ管理（ＤＭ）に対して命令，データ，確認等のやりとり，他のサイトのＴＭとのやりとりを行なう。

(3) データ管理（ＤＭ）

実際のデータベースを管理する（実際の読み込み，書き込み）。

サイトは上の３つの構成要素からなる計算機システムと定義される。

網（ネットワーク）とは，このサイト間の通信システムのことである。

全体の図を図2.5に示す。

トランザクション処理モデルではトランザクションの性質として次のものが必要とされる。

① 原子性（atomicity）

トランザクションはすべてが実行されるかされないかのどちらかである。トランザクションが失敗した場合，それは行なわれなかったものとし，データベースを元の状態に戻すことが必要である。システムがトランザクションを正常終了させることをコミットという。

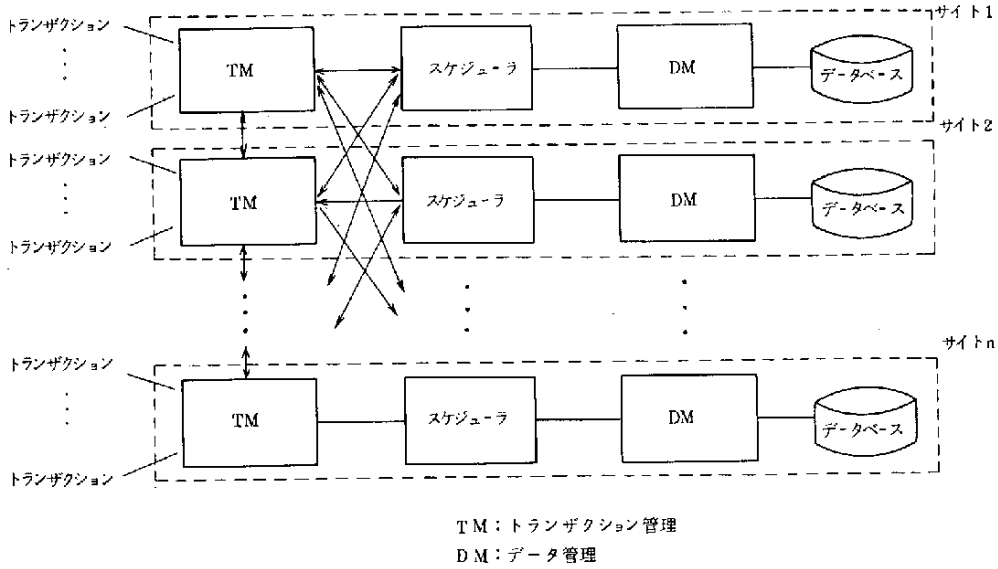


図 2.5　分散型データベースシステムの構成図

② 堅牢性（ Durability ）

一度トランザクションがコミットされたら，その操作による結果が失なわれないことを保証する。この堅牢性を維持することをデータベース回復（リカバリ）という。

③ 遂次性（ Serializability ）

いくつかのトランザクションが実行されている時，その結果はそれらのトランザクションがある順番で遂次に実行されたのと同じ効果を産むこと。同時実行制御の役割はこの遂次性を保証するものと考えられる。

④ 孤立性（ Isolation ）

トランザクションの途中の結果を他のトランザクションが参照できないこと。

⑤ 無矛盾性（ Consistency ）

トランザクションは無矛盾の状態から始まり，終了後も無矛盾性を保つこと。

2.2.2 同時実行制御機構

同時実行制御機構を考える前に，前節（ 2.2.1 ）の図 2.5 にそって各構成

要素の内部動作を次のように抽象化する。

トランザクションは，理想的な形として始まりにはBegin（－transaction），終わりにはEnd（－transaction）をおき，間にはRead，Writeの列を含んでいる（他にも操作における基本要素があるが同時実行制御には関係のないものとして省略する）。TMはトランザクションを受けとると，スケジューラにRead, Writeを送る。スケジューラは同時実行制御アルゴリズムに従って動く。すなわち，DMに送るRead, Writeの列の送出を制御し，

① すぐに出力するか

② それらを保って遅らせるか

③ 取り消すか

の3つの選択を行なう。ここで大事なことは，先のトランザクションの5つの仮定である。DMはスケジューラからRead, Writeを受けるとすぐに実行する。Read の場合は自分のサイトのデータベース内を見に行って要求のあった値を返す。DMがスケジューラに値や確認を返すと，スケジューラはそれをTMに継ぎ渡す。DMは命令の順番に一切関知しない。ここではスケジューラとDM間のやりとりをハンドシェイキングを用いることを仮定する。例えばスケジューラがRead（X）の後にWrite(X）を行ないたいとすると，まずRead（X）をDMに送り，DMの応答（値を返す）を待ち，応答が戻ってから，Write(X)を送り，確認を待つという具合である〔図2.6〕。

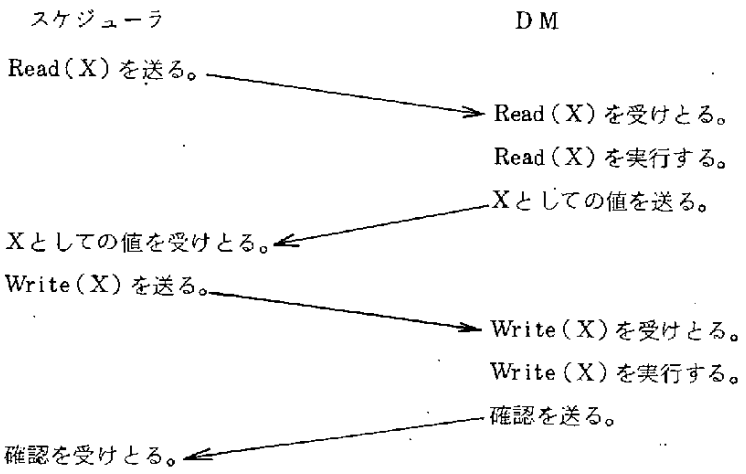


図2.6　DM－スケジューラ間のハンドシェイキング

2.2.2.1　直列性理論（Theory of Serialigability）

同時実行制御方式の正しさを議論するために，ここではその基準となる直列性理論について解説する。

(1)　単一データベースシステムの直列性

問題を単純化するために，単一データベースシステムにおける直列性を議論し，更に分散型データベースシステムに拡張する。単一データベースシステムは単一のサイトが他のサイトと独立したものを仮定すればよい。

まず，操作の単位として読み込みと書き込みを次の形で書く。

$R_i(x)$……トランザクションT_iがデータxを読む。

$W_i(x)$……トランザクションT_iがデータxを書く。

xの単位としてはここでは意識しない。あるトランザクションT_iの読み込み，書き込みの集合をread-set$_i$，write-set$_i$とし，それぞれの集合は必ず共通部分をもつ（disjointでない）ことを仮定する。

スケジューラとは複数のトランザクションらによって実行される操作の列であり，スケジューラによって生成される。

例.

S1：$R_i(x)\ R_j(x)\ W_i(y)\ R_k(y)\ W_j(x)$

トランザクションT_iとT_jがスケジュールSにおいて順序的であるとは，T_iの最後の操作がT_jの最初の操作より前にあること（又はその逆）。そうでない時は同時的であるという。スケジュールが順序性をもつとは，そのスケジュールがどのトランザクション同志も順序的であること，即ち，全順序づけられていることである。例えば上のS1はT_iとT_j，T_kとT_jが同時的であり，順序性をもっていない。

S2：$R_i(x)W_i(x)\ R_j(y)\ R_j(x)W_j(y)\ R_k(y)W_k(x)$

S2は順序性をもっている。トランザクションの順序のみに注目して考えたい場合の記法として順序性をもつスケジュールSに対して，Serial（S）を用いる。上のS2では

Serial（S2）$=T_i\ T_j\ T_k$

である。

操作やトランザクションの順番として，次のように書く。

O_i,O_j：操作　T_i,T_j：トランザクションとすると，

$O_i < O_j$ … O_i はO_j よりも先行する。

$T_i < T_j$ … スケジュールSが順序性をもち，Serial(S) においてT$_i$ がT$_j$ に先行する。

Sが順序性をもたない場合，Serial(S)は定義されず，トランザクションの順序も定義されない。スケジュールの順序性は各トランザクションが正しく実行されるための基準となる。スケジュールが正しく実行される視点から，スケジュールの等価性を定義し，順序性をもつスケジュールと等価なものを直列性(Serialigability)をもつスケジュールとして定義する。

次の2つの条件を満たす時，2つのスケジュールは等価であるという。

① 各読み込み操作は，双方のスケジュール中の同じ書き込み操作で作られたデータを読む。

② 各データの最後の書き込み操作は双方のスケジュールにおいて同じものであること。

この2つの条件は，2つのスケジュールにおいて，読み込み操作において同じ値を読み，結果として，同じ値を書くことを保証している。したがってこの2つのスケジュールが終了した時点で，データベースの状態は同じである。したがって，順序性をもつスケジュールと等価な直列性をもつスケジュールは各トランザクションが正しく実行されることを保証している。

競合の概念を用いても等価性の定義が行なえる。2つの操作が競合(conflict)であるというのは，その2つの操作が同じデータを扱っている異なるトランザクションのものであり，少なくともそのどちらかが書き込み操作である　である。例えば，

$<R_i(x),W_j(x)>$　$<W_i(x),W_j(x)>$は競合

$<R_i(x),R_j(y)>$　$<W_i(x),W_j(y)>$は非競合

である。スケジュールS1，S2が等価であるとは，S1においてO$_i$ がO$_j$ に先行するような各競合組O$_i$ ，O$_j$ について，S2においてもO$_i$ がO$_j$ に先行する。つまり等価の意味は，互いに競合する演算は実行順序により結果が異なってしまうが，競合しない演算はどのような順序で実行されてもよいことに基づいている。例えば，

S3 : $R_i(x)\ R_j(x)\ W_j(y)\ W_i(x)$

S 4 ：$R_j(x)\,W_j(y)\,R_i(x)\,W_i(x)$

において，競合組は＜$R_j(x)$,$W_i(x)$＞これはS 3，S 4どちらにおいても$R_j(x)$＜$W_i(x)$であるから，この2つのスケジュールは等価であり，S 4は順序性をもつので，S 3は直列性をもったスケジュールといえる。

(2) 分散型データベースシステムにおける直列性

DDBSにおいては，各トランザクションはいくつかのサイトにまたがって実行される。したがって各サイトはローカルな（そのサイト内の）スケジュールを持つ。これをローカルスケジュールという。各ローカルスケジュールの直列性を言うだけでは，トランザクションが正しく実行されるとは言えず，次のような条件が必要となる。

DDBSのトランザクションT_1，…，T_nが正しく実行されるとは，

① 各ローカルスケジュールS_kが直列性をもつ

② T_1，…，T_nにおいて，各操作はある統一された順番をもつ。すなわち，もし統一された順番においてT_i＜T_jとならなければならない。

これは，競合の概念を用いて，次のようにも表わせる。

T_1，…，T_n：トランザクション

E：スケジュールS_1，…，S_mでモデル化されたこれらのトランザクションの実行

とすると，

Eが正しい（直列）とは，トランザクションの全体の順番が存在し，各競合組＜O_i,O_j＞が，すべてのスケジュールS_1，…，S_mに対して，それにならった順番である時である。これは，各スケジュールにおいてT_iとT_jの順番の統一がとれていることと一致する。

2.2.2.2 2相ロッキング方式

最初に単一データベースシステムにおける2相ロッキング方式について述べる。

2相ロッキング（2PL）のスケジューラは次の3つの規則により定義される〔Eswaran 7 6〕。

① $R_i(x)$（又は$W_i(x)$）を出す前に読み込みロック（又は書き込みロ

ック）をxに対して設定する。このロックはDMに対して読み込み操作が実行されるまで保持されなければならない。

② 異なったトランザクションは同時に競合ロックを保持できない。2つのロックの競合とは，それらが同一データに対してかかっており，少なくとも1つが書き込みロックの時である。

③ トランザクションは1つでもロックを解いたならば，トランザクションは二度とロックはかけられない。

規則③がいわゆる2相の作法と呼ばれるもので，スケジュールは

・昇相：ロックをかける相

・降相：ロックをはずす相

の2つに分離される。規則③を実現するにはよく，トランザクションが終了するまで，すべてのロックを保持することで行なわれる。すなわちEnd_i（トランザクションT_i のEnd要素）が来た時点でロックをはずす。規則②を守るには，競合するロックを持つトランザクションの操作を遅らせることで行なうが，そうするとデッドロックを導く可能性がある。例えば$R_1(x), R_2(y)$に対して，読み込みロックをかけた後，$W_1(y), W_2(x)$がくると，T_1 はxに対して読み込みロックをかけており，$W_2(x)-T_2$ が待たされる。逆にT_2 はyに対して読み込みロックをかけており$W_1(y)-T_1$ が待たされる。デッドロックの検出は待ちグラフ（wait-for graph）により，行える。トランザクションが他のトランザクションを待つ関係を$T_i \rightarrow T_j$ のように矢印（有向グラフ）で表わすことができ，作られたグラフがサイクルになる時デットロックとなる。例えばT_i がT_j を待ち，T_j がT_i を待つならば，

$$T_j \circlearrowright T_i$$

となり，サイクルができ，デッドロックとなる。

分散型データベースシステムの場合は，2相ロッキングは規則①，②は各サイトで独立に処理できるが，規則③は各サイトのスケジューラとTMの間のやりとりが必要となる。すなわち，トランザクションT_i が終了するにあたって，各サイトのTMはロックをはずす（End_iを送る）前に，T_iの読み込み及び書き込みの完了がDMより確認されるまで待ち，各サイトのTMがT_i のすべての読み込み，書き込みの終了を知った時点で，ロックをはずす（End_iを各スケジューラに送る）。これを行なうには，各サ

イトのＴＭに主となる調整役（ coordinator ）を設け，各サイトのスケジューラからの確認を受けて，トランザクションを終了させる方法がよく用いられる。

ＤＤＢＳの場合はさらに，サイトにまたがったデッドロックの可能性がある。これを解消する方法としては，各ローカルサイトにおける待ちグラフを合成があげられる。

２相ロッキング方式は，単一ＤＢＳにおいて直列性を持つスケジュールを作り出すことにより，その正当性が証明されており〔 Eswaran７６ 〕，ＤＤＢＳの場合も上のように規則③を適用すれば，スケジュールの直列性を証明できる。

2.2.2.3　時刻印（タイムスタンプ）方式

時刻印方式では，トランザクションの一意の時刻印をつけ，その時刻印により，トランザクションが実行される。分散型データベースシステムを仮定する上で，事象の時間的関係を表わす時刻システム〔 Lamport78〕を導入する。

ＤＤＢＳにおける任意の事象Ａ（ 例えばトランザクションＡ ）に対する時刻印をＴＳ（Ａ）と書く。ＴＳ（Ａ）は次のような条件を満たすものと定義される。

① ＴＳ（Ａ）はＡを一意に識別する。
すなわち異なった事象は異なった時刻印がつく。

② 事象Ａ，Ｂに対して，Ｂの前にＡが起こるとは，時刻印の関係としてＴＳ（Ａ）＜ＴＳ（Ｂ）の時である。これをＡ→Ｂと表わす。

分散型のシステムを仮定した時，前に起こるという関係を次のように定義する。

ⓐ もしＡ，Ｂが同じサイト上の事象で，時間的にＡがＢの前に起こる時，Ａ→Ｂ

ⓑ もし，事象Ａがメッセージを送り，Ｂがそのメッセージを受けるならば　Ａ→Ｂ（サイトをまたいでもよい）

ⓒ もし，Ａ→Ｂ，Ｂ→ＣならばＡ→Ｃ

事象Ａ，ＢがＡ→ＢでもＢ→Ａでもない時，ＡとＢは仮同時的であるという。例えば図2.7ではＡとＤ，ＢとＥ，ＢとＦ，ＣとＦはそれぞれ仮同

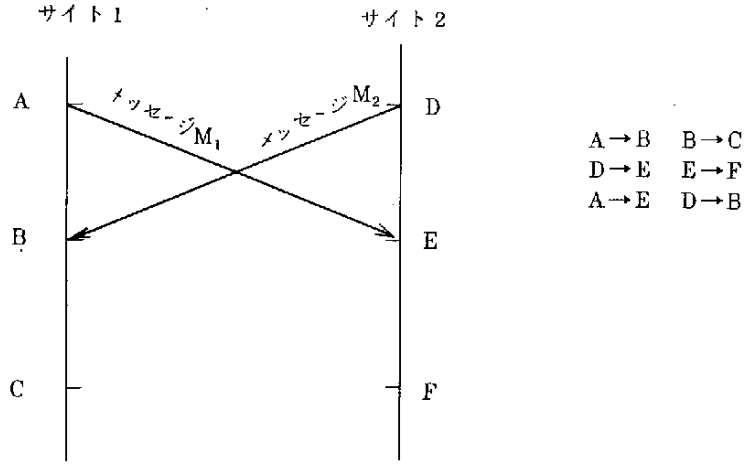


図 2.7　サイト間の事象とメッセージ

時的である。

時刻印の定義により，2つの事象A，Bに対して，A→BならばTS(A)＜TS（B）である。分散型のシステムの時刻印の付けかたとしては，サイト識別子とカウンタを用いる方法がある。すなわち，時刻印として，カウンタCとサイト識別子Sの組＜C，S＞を用いる。カウンタはサイト内の事象の順番を示し，大きいほど新しい事実とする。又，サイト識別子はサイトに一意につけられた番号である。時刻印のつけかたとしては次の3つの規則を用いる。

① サイト内の事象の時刻印のサイト識別子はそのサイトの番号がつく。

② サイト内のメッセージの送られてない事象は直前の事象の時刻印のカウンタを1つ増やしたものとする。

③ 別サイトの事象（これをAとする）からメッセージを受ける事象（これをBとする）の時刻印はBがメッセージを受けてない時の時刻印（これを本来の時刻印とする）のカウンタとAの時刻印のカウンタを比べ，Bのカウンタが Aのカウンタよりも大きい時は，本来の時刻印とし，そうでない場合はAの時刻印のカウンタを1つ増やしたものとし，サイト識別子はそのサイトの番号とする。

時刻印による同時実行制御は大きく分けて，基本型時刻印方式と保存型時刻印方式の2つの方法がある。

A．基本型時刻印方式

基本型時刻印方式は次の規則に従って，各サイトで行なわれる。

① 各トランザクションは前述した時刻システムに従って時刻印をもつ（通常ＴＭが行なう）。

② 各トランザクションの操作（ReadとWrite)はそのトランザクションの時刻印をもつ。

③ それぞれのデータ x に対して，今までの読み込み操作の中で一番大きな（若い）時刻印としてＲＴＭ（x），書き込み操作の中で一番大きな時刻印ＷＴＭ（x）を内部に持つ（通常スケジューラ内）。

④ 今，x に対する読み込み操作の時刻印をＴＳとする。もしＴＳ＜ＷＴＭ（x）(読み込みが古い)ならば，その読み込み操作を取り消し，それを発行したトランザクションを新しい時刻印をつけて再起動する。そうでなければ読み込みを実行し，必要ならば，③に従って，ＲＴＭ（x）を更新する〔図2.8〕。

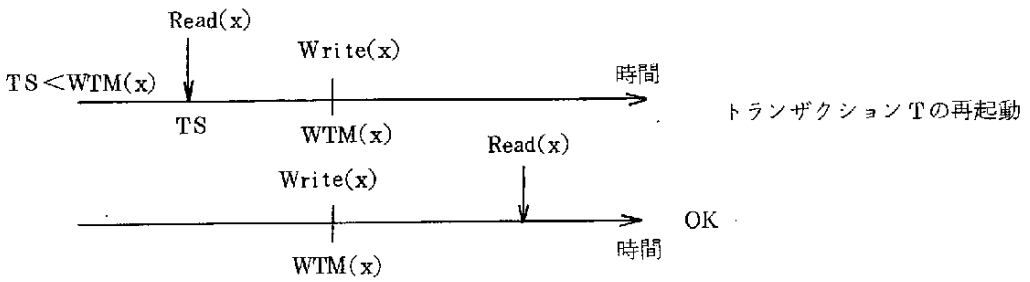


図2.8　時刻印方式による読み込み規則

⑤ 今，x に対する書き込み操作の時刻印をＴＳとする。もしＴＳ＜ＲＴＭ（x）又は，ＴＳ＜ＷＴＭ（x）の時（書き込みが古い），その書き込み操作を取り消し，それを発行したトランザクションを新しい時刻印をつけて再起動する。そうでなければ書き込みを実行し，必要ならば③に従ってＷＴＭ（x）を更新する〔図2.9参照〕。

時刻印方式では④，⑤が競合する操作に対する処理になっており，統一したトランザクションの順番が保たれることになる。

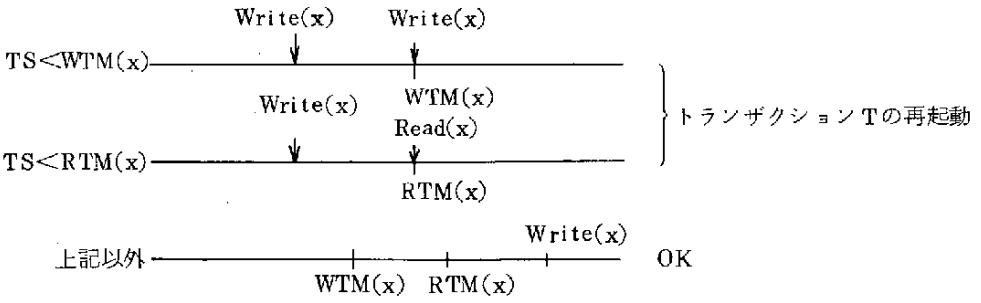


図 2.9　時刻印方式における書き込み処理

B．保存型時刻印方式

保存型時刻印方式では，各サイトは他の全てのサイト毎にバッファを持ち若いトランザクションの操作が古いトランザクションの操作と競合する心配がなくなるまで操作を，バッファに保存しておく。次の規則に従って各サイトで行なわれる。

① 各トランザクションはあるサイトでのみ実行され（管理され），他サイトへは読み込み，書き込み操作を要求するのみとする。

② サイト i は異なったサイト j からの読み込み，書き込み要求を時刻印の順番で，受けとる必要がある。

③ サイト i は，各サイトからの読み込み，書き込み操作を少なくとも1つバッファ内に持つ時，サイト i は次のようにふるまう。

a．サイト i に届く読み込み操作Rに対して，もし，ＴＳ（Ｒ）＞ＴＳ（Ｗ）なる書き込み操作Wがサイト i のバッファにあれば，これ（ら）の書き込み操作が実行されるまで，Ｒはバッファに入れられる。そうでなければＲは実行される。

b．サイト i に届く書き込み操作Wに対して，ＴＳ（Ｗ）＞ＴＳ（Ｒ）であるような読み込み操作Ｒ，又はＴＳ（Ｗ）＞ＴＳ（Ｗ′）であるような書き込み操作Ｗ′がバッファ中にあれば，これ（ら）の操作が実行されるまでバッファに入れられる。そうでなければ，Ｗは実行される。

この方式を図式化したものを図２．１０に示す。

保存型時刻印方式ではあるサイトが１つの操作も他のサイトに送らない時，③の最初に述べた仮定が満たされない。そこで，各サイトは他のサイ

トに，時刻印のついた空操作を定期的に送ることがよく行なわれる。

時刻印による同時実行制御は統一されたトランザクションの順番を保証し，直列性をもつスケジュールを作ることができる。

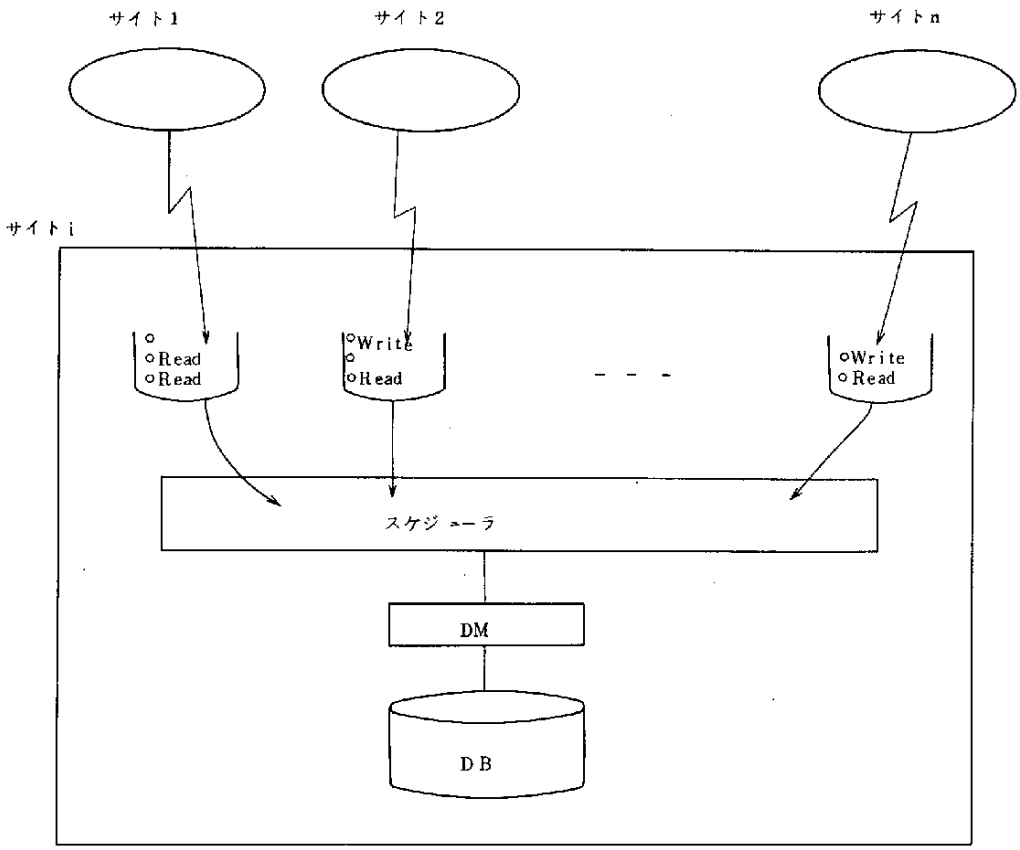


図2.10

2.2.3 信頼性制御方式

信頼性の制御の目的は，種々の障害に対して，データベース全体を無矛盾な状態に保つこと，すなわち，トランザクションの原子性（atomicity）を保証することにある。DDBSを考える場合，トランザクションの操作は各サイトにふりわけられ，各サイトのTMにより，実行される。今，システム

全体の回復を考える前に，各サイトでの信頼性を保証することを考える。これは単一ＤＢＳの議論をそのまま使うことができる。

2.2.3.1　単一データベースシステムの回復

単一ＤＢＳにおける障害（ failure ）と回復（ recovery ）方法を考える。

(1)　単一ＤＢＳにおける障害のモデル

障害における重要な点は情報が失なわれるかどうかということである。これは主に４つのタイプに分けて考えることができる。

①　情報の損失がない場合

主記憶上の情報が使用可能であるような障害である。例えば，桁あふれとか，零割り算などがこのタイプに入る。

②　主記憶上の情報のみが失なわれる障害

ディスク上に記録された情報は残っている。この種の障害はシステム故障があげられる。

③　ディスク上の情報が失われる障害

媒体故障。ディスク上の情報が失われる。この種の障害にヘッド破壊があげられる。これを軽減するためにディスクのコピー版を持つような堅牢な記憶媒体を用いる。

④　堅牢な記憶媒体の情報が失なわれる障害，確率はかなり低くなるが，まったく０というわけにはいかない。

(2)　ログ（ Logs ）

ログは，トランザクションにより実行されたすべての動作を再施行（ redo），やり戻し（ undo）するための情報を含んでいる。

トランザクションのやり戻しとは，そのトランザクションの実行前のデータベースの状態に戻すことである。トランザクションの再施行とはもう一度そのトランザクションをやり直すことである。

やり戻し，再施行するのはトランザクションが失敗した時に原子性を保つために必要である。やり戻しは，トランザクションが失敗した時，それまでの結果をやり戻しにより，帳消しに必要がある。又，再施行はその逆で，障害のため完了していないトランザクションをもう一度やり直すために必要である。

ログには基本単位としてログレコードが通常含まれ（普通各トランザ

クションの操作ごと）ており，主に次の情報により構成されている。

① トランザクションの識別子

② ログレコードの識別子

③ 動作のタイプ（挿入，削除，更新等）

④ 操作の対象となるデータの古い値

⑤ 操作の対象となるデータの新しい値

⑥ 回復手続きに必要な補助情報

（同じトランザクションの前のログレコードへのポインタ等）

他の種類のログとして，データの古い値と新しい値の変化分だけを記録する方法（遷移ログ）もある。上記の方法はこれに対して状態ログともいう。

さらにトランザクションの起動時，コミット時，取り消し時に各々Begin，Commit，Abortに対するレコードがログに書かれる。

ログにログレコードを書くタイミングは，データベースが更新する前に堅牢な記憶装置に書かれていることが必要である。したがって次のような規則（ログ先書きプロトコル）を用いる。

① データベースを更新する前に，少なくとも対応するログレコードのやり戻し用の部分が堅牢な記憶装置に記録されていること。

② トランザクションがコミットされる前にトランザクションのすべてのログレコードは堅牢な記憶装置上に記録されていること。

(3) サイトの障害に対する回復手続き

主記憶上の情報が失なわれる障害の場合，回復手続きはログファイルを読み次の操作を実行する。

① やり戻しされるべきすべてのコミットされていないトランザクションをみつける。（ログにコミットのレコードがないことでわかる）

② 再施行される必要のあるすべてのトランザクションをみつける。これらはログにおいてコミットのレコードがあるものである。

③ ①でみつけたトランザクションをやり戻す。②でみつけたトランザクションを再施行する。

①と②で，やり戻し，再施行されるトランザクションを限定するためにチェックポイントを設ける。チェックポイントは定期的に行なわれる操作で次の3つのステップから成る。

a．堅牢な記憶装置上にログレコードを書く。その間データベース上の更新内容は主記憶上にある。

b．堅牢な記憶装置のログ上にチェックポイントレコードを書く。チェックポイントレコードには，その時点で動作しているトランザクションの識別子が入る。

c．主記憶内の更新されるデータを，全て2次記憶に書き出す（強制書き込み）。

チェックポイントの重要な概念は，チェックポイント時には主記憶のバッファと2次記憶内のデータの値が一致していることである。チェックポイントを用いると，上の回復手続きの①，②は次のようになる。

イ．最後のチェックポイントをみつける。

ロ．チェックポイントレコード中のトランザクションをやり戻しの対象とする。再施行の対象はこの時空とする。

ハ．ログファイルをチェックポイントレコード以降から最後まで読む。この時Beginレコードがあれば，それをやり戻しの対象とする。コミットレコードがあれば，それに対応したトランザクションをやり戻しの対象から再施行の対象に変える。

堅牢な記憶装置上の情報が失われる障害に対する回復の場合，2つのことが考えられる。

① データベースの情報が失われ，ログが残っている場合

② ログも失われた場合

①の場合はあらかじめデータベースがダンプ（dump)されていれば再施行により元に戻すことができる。②の場合は元に戻すことは不可能で，ダンプをとった時点までしか復元できない。

2.2.3.2 分散型データベースシステムの回復

分散型データベースシステムでは通信における障害の問題がある。通信における障害としては多種のものが考えられるので，問題を単純化するため，次のことを仮定として要求する。

メッセージがサイトXからサイトYへ送られるとした時，次のことが必要である。

① Xはある遅れの最大値ＤＭＡＸ以内に，Yからの受信確認を受けと

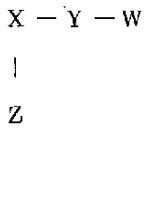
る。

② XからYへのメッセージは正しい順番でYに到着する。

③ メッセージは正しい。

④ XがYからの確認を受けたら，メッセージはYに必ず到着したものとする。

通信の障害は，網の分割により抽象化される。すなわちサイト間の通信路が切れることにより，網がより小さいグループに分割される。例えば図2.11において，サイトXとサイトYの間が切れた場合，網システムは図2.12のようにより小さなグループに分割される。この時ZとY，ZとWも通信が不能となる。



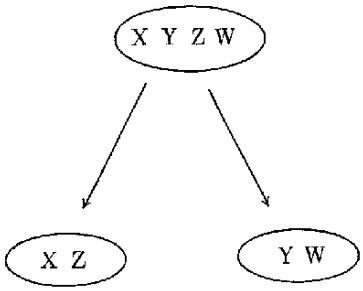


図2.11　サイト間のネットワーク　図2.12　ネットワーク分割

サイトXがサイトYにメッセージを送った後，ある待ち時間をおいても確認が返ってこなかった時，次のことが不明である。

① 障害は起こったのか，それともシステムが遅いのか？

② 障害が起こったとして，通信の障害か，サイトYの障害か？

③ メッセージはYに到着したのか？

DDBSの場合，単一サイトの障害に加えて，上述した網の障害を含め，どこまでの障害を仮定するかにより，回復の手順が決まる。

クラス1．サイトの障害のみ。

クラス2．サイトの障害及びメッセージの紛失，網分割なし。

クラス3．サイトの障害及びメッセージの紛失，網分割あり。

DDBSでは各サイトのトランザクション管理－TMがローカルな回復を行なうことにすれば，単一データベースにおける回復の議論が可能とな

る。つまり，トランザクションを各サイトで行なわれるものに分割し，トランザクションの体裁を調えるように，Local-begin，Local-commit，Local-abort等の基本要素をつけ加えたものをサブトランザクションとする。

ＤＤＢＳの場合は，サブトランザクションの原子性が保たれる必要があるだけでなく，次の条件が必要となる。

① 各サイトのサブトランザクションは行なわれたか，まったく行なわれなかったかのどちらかである。即ちサブトランザクションの原子性の保障が必要となる。

② すべてのサイトは，サブトランザクションのコミット又は取り消し（ abort ）に関して，同じ決定をする。

②を保障するには，ＴＭ中で調整役（ coordinator ）となるものを決め，この調整役がトランザクションの最終的なコミットあるいは取り消しを行なう方法がある。

各サイトがトランザクションに対して同じ決定（コミット又は取り消し）を行なうための制御機構をコミットメント制御（プロトコル）と呼ぶ。

2.2.3.3　2相コミットメントプロトコル

2相コミットメントプロトコルは2種類のＴＭのタイプにより行なわれる。

調整役……1つのＴＭで最終的なコミットあるいは取り消しを行なう。（coordinator）

参加者……調整役でない他のサイトのＴＭで，ローカルＤＢ上での操作をＤＭに送る。これは各サイトに必ず1つ置かれる（調整役のあるサイトにも含まれる）。（participants）

2相コミットメントは2つの相（フェーズ）に分かれる。

フェーズ1.　コミットメントの準備

各参加者は，そこでのサブトランザクションが終了する段階で，準備完了（ ready ）を調整役に送る。調整役はコミットメントのメッセージを送

る前に，準備（ prepare ）ログレコードを堅牢な記憶装置に書き，＂準備＂メッセージを各参加者に送る。＂準備＂メッセージには，サブトランザクションの識別子が入っている。参加者が準備完了を送ると，そのサイトではサブトランザクションをコミットできることを意味している。そこで，各参加者は堅牢な記憶装置に２つのものを書き込む。

① サブトランザクションをローカルにコミットするために必要なすべての情報

（サブトランザクションのすべてのログレコード）

② ＂準備完了＂ログレコード

調整役はコミットするか，取り消すかを参加者から送られてきた＂準備完了＂又は＂取り消し（ abort ）＂，もしくは時間切れ（ time-out ）より判断する。

すなわち，すべての参加者が＂準備完了＂を返答した場合，トランザクションをコミットと判断する。少なくとも１つの参加者が＂取り消し＂，もしくは＂時間切れ＂を返した場合は，トランザクションを取り消しと判断する。

フェーズ2.

調整役はフェーズ１で判断した結果により，まずログに＂全体コミット（ global-commit ）＂又は，＂全体取り消し（ global-abort ）＂レコードを堅牢な記憶装置に書き込む。その後，各参加者にその決定を通知する。すべての参加者は調整役の通知により，＂コミット＂もしくは，＂取り消し＂レコードをログに書き込む。この時，＂取り消し＂ならば，回復を行なう。最後に，すべての参加者は最終的な確認（ＡＣＫ）メッセージを調整役に送る。調整役はすべての参加者から確認メッセージを受けると＂完了（ complete ）＂レコードをログに書く。

図2.１３に２相コミットメント制御のプロトコルを示す。

２相コミットメントプロトコルはログレコードの紛失がないことを仮定して，様々な障害に対する対処が考えられる。

① サイトの障害

ａ．参加者がログに＂準備完了＂レコードを書く前に障害が起った場合

→時間切れとなり調整役は取り消しと決定する。

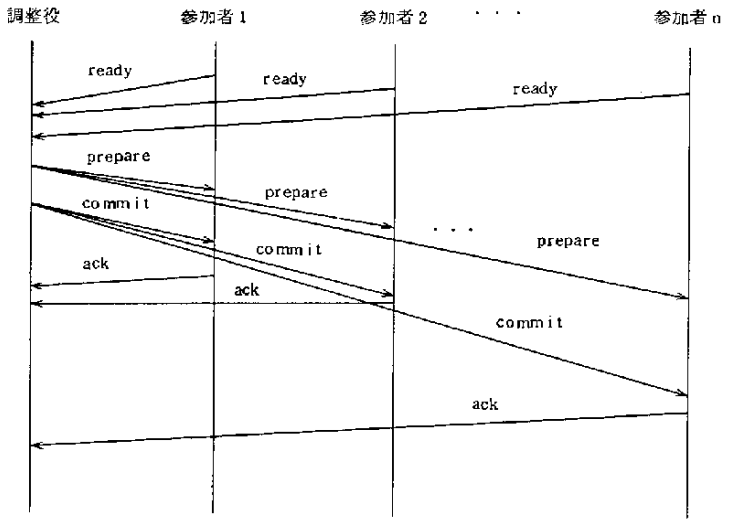


図2.13　2相コミットメント制御のプロトコル

b．参加者が，ログに"準備完了"レコードを書いた後に障害が起った場合

→調整役に障害が起った旨のメッセージを送り，トランザクションの成り行きを問い合せ，適当な動作（コミット又は取り消し）を行う必要がある。

c．調整役が"準備"レコードを書いた後に障害が起った場合（ただしまだ"全体コミット"，"全体取り消し"レコードをログに書く前）。

→調整役が回復を行ない，改めて，"準備"メッセージを送る。

d．調整役が"全体コミット"又は"全体取り消し"レコードをログに書いた後，障害が起った場合（但しまだcompleteレコードは書いていない。

→調整役はあらたに決定を行ない（コミット又は取り消し），参加者に通知する。

e．調整役が"完了"レコードを書いた後に障害が起った場合。

→すでにトランザクションは終了している。

② メッセージの紛失

a．参加者からの"準備完了"又は"取り消し"メッセージを紛失した場合

→時間切れで，調整役は取り消しと判断する。

b．調整役からの"準備"メッセージを紛失した場合

→参加者は待ったままで，調整役に返答を返さない。時間切れで取り消しと判断する。

c．参加者からのコマンドメッセージ（"コミット"又は"取り消し"）が紛失した場合

→参加者からの応答がないため時間切れとなり，再びメッセージを送る。

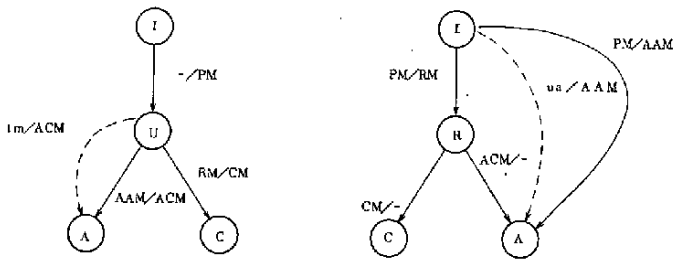
d．参加者からの確認メッセージが紛失した場合

→cと同じ。

③ 網分割

2つのグループ，調整役を含むグループと含まないグループに分かれたとすると，調整役を含むグループでは含まないグループのサイトの障害に見える（①a，①b）。含まないグループでは調整役のサイトの障害に見える。

以上まとめると2相コミットメントプロトコルは状態遷移図で表現できる（図2.14参照）。



調整役

I：初期状態
U：不確定状態
R：コミット準備状態
A：取り消し状態
C：コミット状態

参加者

RM　："準備"メッセージ
RM　："準備完了"メッセージ
AAM："取り消し"返答メッセージ
ACM："取り消し"コマンドメッセージ
CM　："コミット"コマンドメッセージ
tm　：時間切れ
ua　：サイト独自の取り消し

→メッセージによる推移
⇢独自の原因による推移
α / β　αは来るメッセージ又は独自の原因（tm又はua）
βは生成されるメッセージ

図2.14　2相コミットメントプロトコルの状態遷移図

2.2.3.4　チェックポイントとコールドリスタート

コールドリスタートは，運用中のデータベース自身の致命的な障害の回復に必要とされる手順である。致命的な障害としては，堅牢な記憶装置中のログの消失が挙げられる。この場合，無矛盾なデータベースの以前の状態まで引き戻さなければならない。前の無矛盾な状態はチェックポイントにより印がされている。ＤＤＢＳの場合は，コールドリスタートの問題は難しい。トランザクションは各サイトにまたがっており，あるサイトが前の無矛盾な状態に引き戻されるなら，他のすべてのサイトも同じ前の状態に引き戻さねばならない。

無矛盾なデータベースの全体状態Ｃは次の２つの特徴をもつ。

① 各トランザクションＴに対して，Ｃは各サイトでのＴのすべてのサブトランザクションの更新を含んでいるか，まったく含んでいないかのどちらかである。含んでいる時ＴはＣに含まれるという。

② ＴがＣに含まれている時，Ｔと競合し，かつＴに先行する全てのトランザクションはＣに含まれる。

①はトランザクションの原子性を保証するものであり，②は，Ｔと競合するトランザクションの結果は，Ｔに影響を与えるために必要である。

全体的に無矛盾な状態を構築するには，次のものを使用する。

- 各サイトのローカルダンプ
- 各サイトのローカルログ
- 全体チェックポイント

全体チェックポイントは，各サイトのローカルチェックポイントの集合である。これらのローカルチェックポイントは次の条件により同期がとられていることが必要である。まず，各サイトi（$i=1, \cdots, n$）のローカルチェックポイントをＬＣiとする。サイトiで実行されるサブトランザクションをＴiとする。ＬＣiにＴiが含まれるとは，ＬＣi以前に，Ｔiが正常終了（コミット）していることとする。ここで，各サイトiのあるローカルチェックポイントをＬＣiとする。ＬＣi内の各サブトランザクションをＴiとし，Ｔiの元となるトランザクションをＴとする。このとき，各サイトiのあるＬＣiについて，ＬＣiに含まれる各Ｔiについて，Ｔiの元のトランザクションＴの各サブトランザクションＴjは，各サイトjのＬＣjに含まれるとき，これらのＬＣ$_1$，…ＬＣn は，同期

がとれているといい，集合｛ＬＣ₁，…ＬＣn ｝を，全体チェックポイントとする〔図2.15.〕。

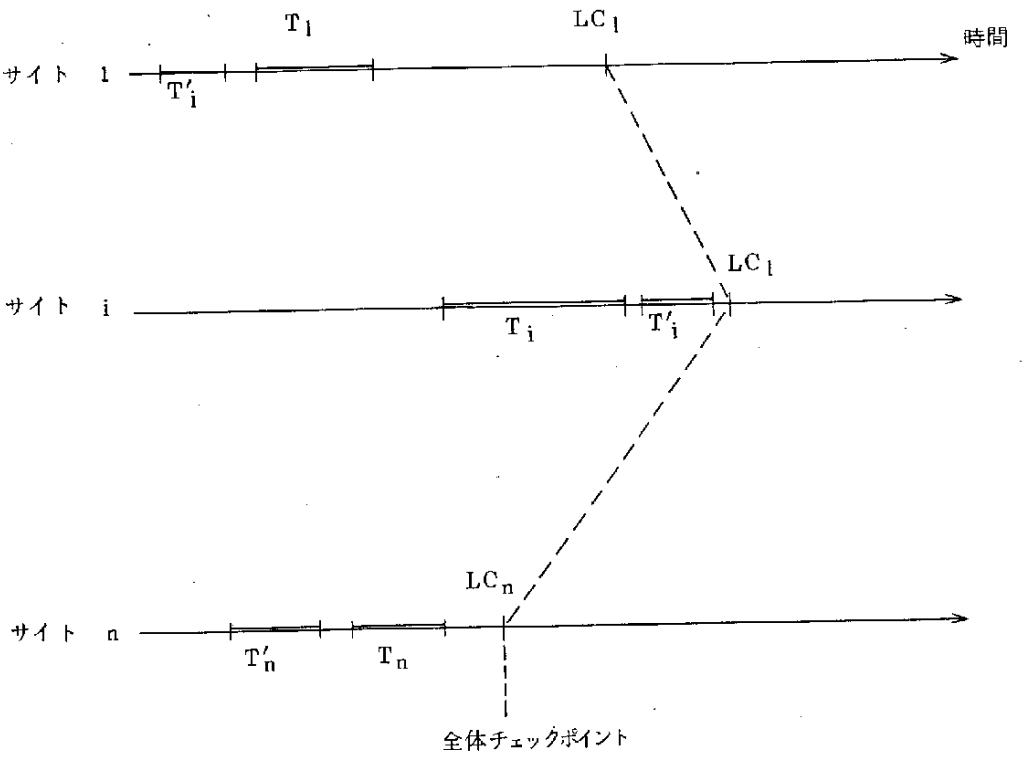


図2.15　全体チェックポイント

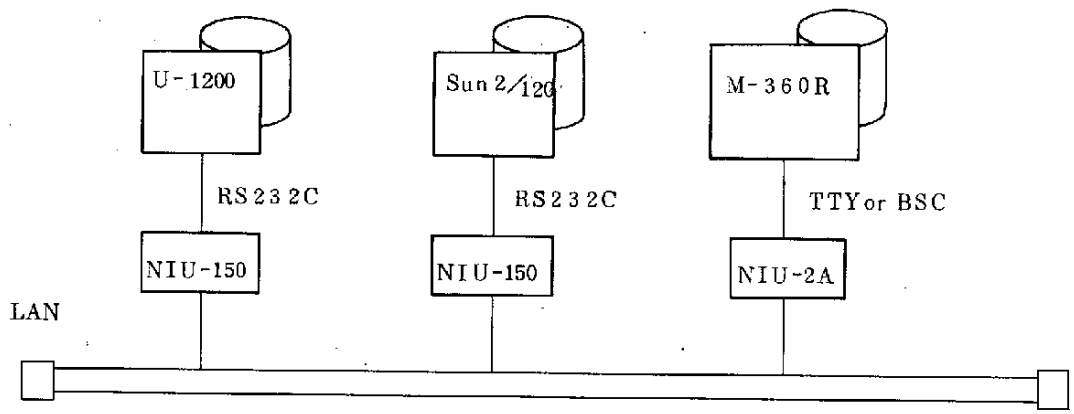
各サイトのローカルチェックポイントの同期をとる方法としては，二相コミット方法により全サイトで合意をとる方法がある。これによると，コミットをとっている間，ＤＤＢＳ全体がブロックしてしまい，システムの有効性（ availability ）を低下させてしまう。これに対して，〔Schlageter８０〕は，より有効ないくつかの方法を提案している。例えば，トランザクションＴが，Ｔが入力されたサイト i の最後のＬＣ i （全体チェックポイントＧＣ）後に起動されたならば，この全サブトランザクション T_j は，サイト j のＬＣ j （ＧＣ）の後に起動される様に制御する方法である。

2.3　ローカル情報システム（ＬＩＳ）のアーキテクチャ

ＬＩＳの実現に関して必要となるネットワークアーキテクチャを中心として，放送網を有効に活用した議論を行なう。

2.3.1　ＬＩＳの全体構成

ローカル情報システム（ＬＩＳ）はネットワークとしてローカルエリアネットワーク（ＬＡＮ）を用いた分散型データベースシステム（ＤＤＢＳ）である。各サイトには汎用計算機及びミニコン，ワークステーションがあり，各ローカルなデータベースシステムの統合化がはかられている。ＬＩＳの実際の構成を図2.１６に示す。



ＬＡＮ	アンガマンバス社　Net／one
	Ethernet 型　CSMA/CD
	ベースバンド　同軸（１０Mbps）
	サービス……Ethernet Data link，バーチャルサーキット，データグラム等
ミニコン	パナファコム社　U－１２００
	１６bit CPU　主記憶1MB
	ディスク　１３０ＭＢ
	ＯＳ　UNIX Ⅲ＋4.2 bsd
ワークステーション	Sun　マイクロシステム社　Sun２／１２０
	ＣＰＵ68010 主記憶２ＭＢ
	ディスク　１４４ＭＢ
	ＯＳ　UNIX 4.2 bsd
汎用計算機	富士通社　Ｍ－３６０Ｒ
	ＯＳ　Ｆ４

図2.１６　ＬＩＳの構成

各サイトのデータベースシステムとしては，Ｍ－３６０ＲではＡＩＭ／ＤＢ（ＣＯＤＡＳＹＬ型）とＡＩＭ／ＲＤＢ（関係型）があり，Sun２／１２０，Ｕ－

１２００上では第４章で述べるGranada(関係型)がある。

ＬＩＳの機能レベルの構成としては，各サイトに仮想情報システム(VIS)があり，ＶＩＳにはワークステーションや超小型計算機上に実現された個人情報システム(ＰＩＳ)と，従来の汎用計算機上に実現された中央情報システム(ＣＩＳ)の２種がある。ＰＩＳとＣＩＳは以下の機能を有している。

ＰＩＳ

① ＰＩＳ内のローカルなデータ管理機能。関係型のＤＢＭＳ機能。

② 他のＶＩＳ間のデータ通信機能。

③ 高水準利用者インターフェース機能。

ＣＩＳ

① ＬＩＳ全体の共有データ，及び従来のデータ管理機能。

② 他のＶＩＳ間のデータ通信機能

各ＶＩＳは，(i)データベースとしてのＤＢと，(ii)ＤＢ内のデータ管理を行なう仮想データベースプロセッサ(ＶＤＰ)と，(iii)ＶＩＳ間の通信処理を行なう全体データベースプロセッサ(ＧＤＰ)から構成される(図2.17参照)。

① ＶＤＰ

ＶＤＰは，利用者にローカルなデータに対する関係型のデータベースシステムとしてのサービスを提供する。大型機のCODASYL型ＤＢＳに対しては，関係インターフェースシステムＬＤＰ〔Takizawa８０，８２〕により，関係型とみせることが可能である。

② ＧＤＰ

ＧＤＰは，ＶＩＳ間でのＤＢＳ間の通信処理を行なうための管理システムであり，以下の機能を有している。

(i) 分散検索演算処理(ＤＱＰ)

利用者にデータの所在を意識させずに，非手続き的なデータ検索を行なわせるための処理

(ii) 分散更新演算処理(ＤＵＰ)

更新演算に対して，各ＤＢＳ間のデータの無矛盾性を保証するためのトランザクション管理(コミットメント制御，同時実行制御，障害回復)。

(iii) 安全性管理

ＤＤＢＳ内のデータへの不正アクセスと，情報の不正流出の防止。

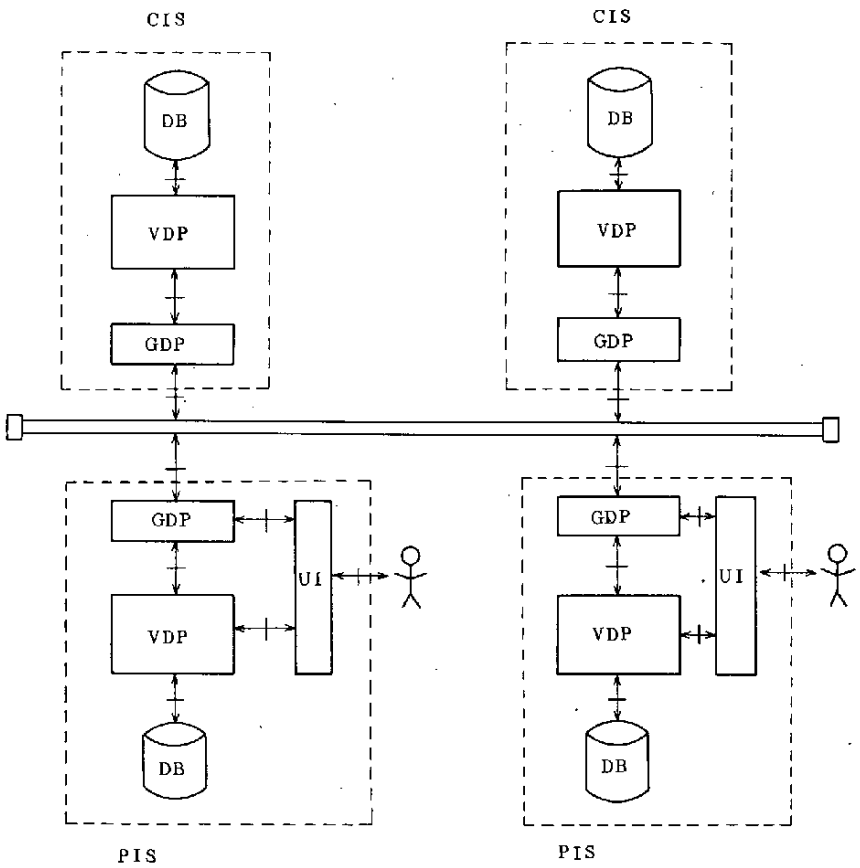


図2.17　ＬＩＳの機能構成

(iv)　ＤＤ／Ｄ管理

ＤＤＢＳ内のデータと応用の所在，操作方法についての管理。

③　ＵＩ

ＵＩは利用者に高水準なデータベース利用を行なわせるインターフェースシステムである。

(i)　関係論理方式の言語（ＲＱＬ）。

(ii)　２次元インターフェース

④　ＬＡＮ

ＬＡＮは，ＶＩＳ間での通信を提供するシステムである。ＬＩＳでは放送通信を行なうため以下の特徴をもつ。

(i) 通信量，通信時間の減少

1回の送信によって，全点に同一の情報が届くことによる。

(ii) 同時到着性

あるＶＩＳで送信された情報は，同時に全点に到着する。

2.3.2 放送網と群サービス

前節で述べたＧＤＰは放送通信の機能を有しており，各種の処理（問い合せ，コミットメント）に対して有効な手続きを与える。

ＧＤＰの通信網としてはEthernet のデータリンク（ＥＤＬ）サービスを提供している。ＧＤＰはＥＤＬサービスを用いて，他のＧＤＰと一体となり，複数のＶＩＳ間の高信頼な放送通信を提供する。ＧＤＰはさらに図2.18に示す階層構造を有している。トランスポート制御（ＴＧ）層は，ＥＤＬサービスを用いて，高信頼な放送通信サービスを提供する。ＤＤＢ層はＴＣサービスを用いて，ＶＩＳ間での分散トランザクション処理を行なう。

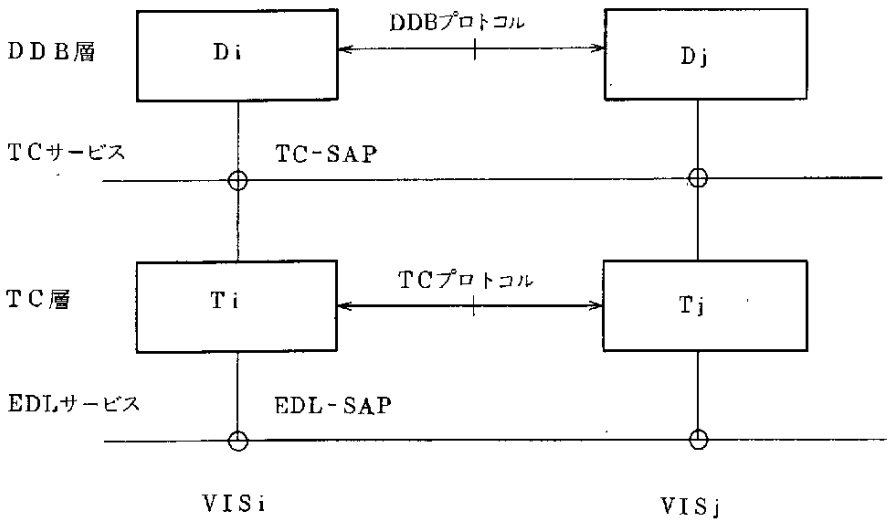


図2.18　ＧＤＰの通信階層

ＴＣサービスにおいて放送通信がサポートされているが，この放送通信の機能としては群サービスを提供している。

群サービス

複数のＶＩＳは群を形成し，あるＶＩＳから送信されたパケットは群内の全点に放送される。群サービスとして次の特徴をもつ。

(i) 動的群と永続的群

- 動的群

  必要な都度，ＶＩＳ間で群を設定する。

- 永続的群

  永久的に設定されている群

(ii) 高信頼性　ＦＩＦＯ性

情報は出した順番に確実に届く。

ＴＣのプロトコルについては次章で詳しく述べられる。

ＤＤＢ層はＴＣサービスの放送性を用いて，各種処理の効率的な手法を有している（問い合せの最適化，コミットメント制御の最適化）。

2.3.2.1　問い合せに対する最適化

分散型の問い合せの手順を与えるものとして分散検索演算処理手順がある。

分散型検索演算処理手順

X_i をVIS_i 内の関係とする（$i=1, \cdots, n$）。このとき$X_1, \cdots, X_n$ を属性aについて結合（join）し，その中で属性集合｛$a_1, \cdots, a_n$｝（a_i はx_i の属性）の値を求める問題を考える。このための通信処理手順〔Takizawa 81，83c〕B^2Cを以下に与える。

〔B^2C手順〕

① ある関係X_1〔a〕をある順序で順序づけて，T_1 とし，T_1 を放送する。

② 各VIS_i は$U_i = X_i$〔a=a〕T_1 ； $T_i = U_i$〔a_i〕；T_1 の各値vに対して，vがT_i 内にあれば，その位置のビットをＯＮにしたビットマップBM_{1i} をつくり，放送する。

③ 各VIS_i は他のVIS_j からBM_{1j}を受信したならば，$BM_{1i} = BM_{1i} \wedge BM_{1j}$ ；すべて受信したならば，T_1 内のB_{1i}のビットがＯＮに対応した値集合Ａを求め，$T_i = (X_i$〔a=a〕A）〔a，a_i〕を利用者

のいるＶＩＳに送信する。

④　ＶＩＳは各T_iをａについて結合し，目標集合Ｔを求める。

図2.19にこの手順例を示す。この時，全通信量は，X_1〔a〕とビットマップのビット量に比例する。これは，一対一通信を用いた従来の方法〔Hevner78〕に対して，全通信量を大幅に低減できる。

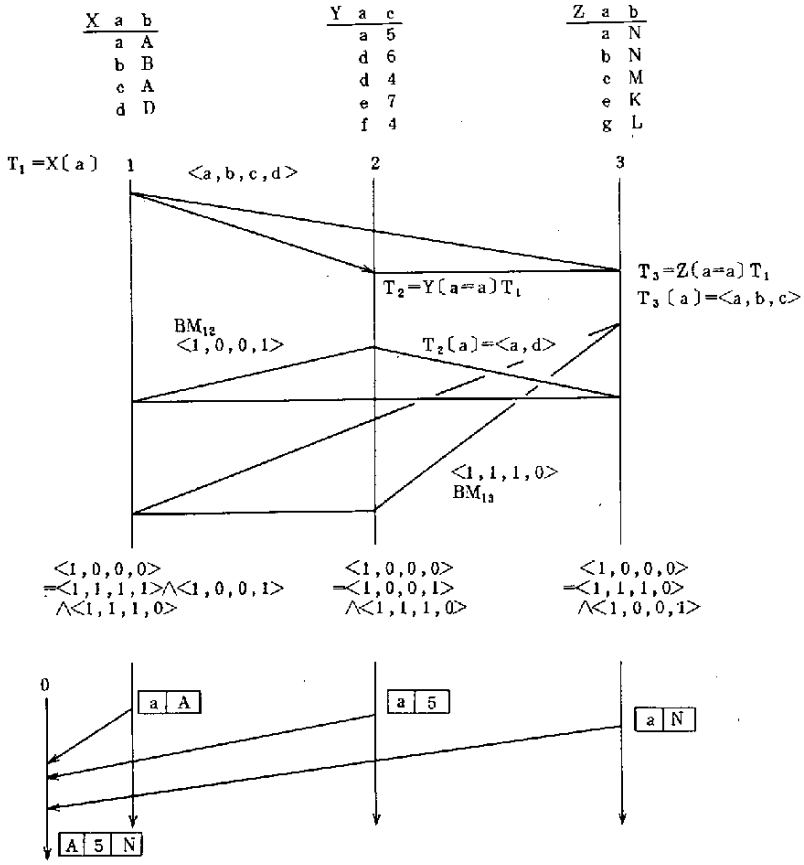


図2.19　B^2C手順例

2.3.2.2　コミットメント制御に対する最適化

前節で述べたトランザクションのコミットメント手順（２相コミット等）の放送通信を用いたものを述べる。

ｎ個のデータx_1，…，x_nを更新するトランザクションをＴとすると，以下の手順によりＴの原子性が保たれる。図2.20にその様子を示す。

① Tから $x_1, \cdots, x_n$ に更新するべきデータを放送する。

② 各 x_i は更新データを自サイトのログに格納し，その後 precommitted（前コミット完了）メッセージを放送する。

③ 各サイトは，他の precommitted を待つ。全て受信すれば，ログ内の更新データを用いて，データベース内の更新を行なう。

この手順は本質的には2相コミットメントと変わらない。2相コミットにおける第1相は①，②が対応し，2相コミットの"準備完了（Ready）"メッセージに対して，"precommittd"がある。第2相は③が対応するが，調整役によって，すべてのサイトの"precommitted"メッセージを集める必要がなく，各サイトが他のサイトの結果をまとめて，独自にコミットすることができる。したがって通常の2相コミットメントの通信回数の約半分の手順ですむ。

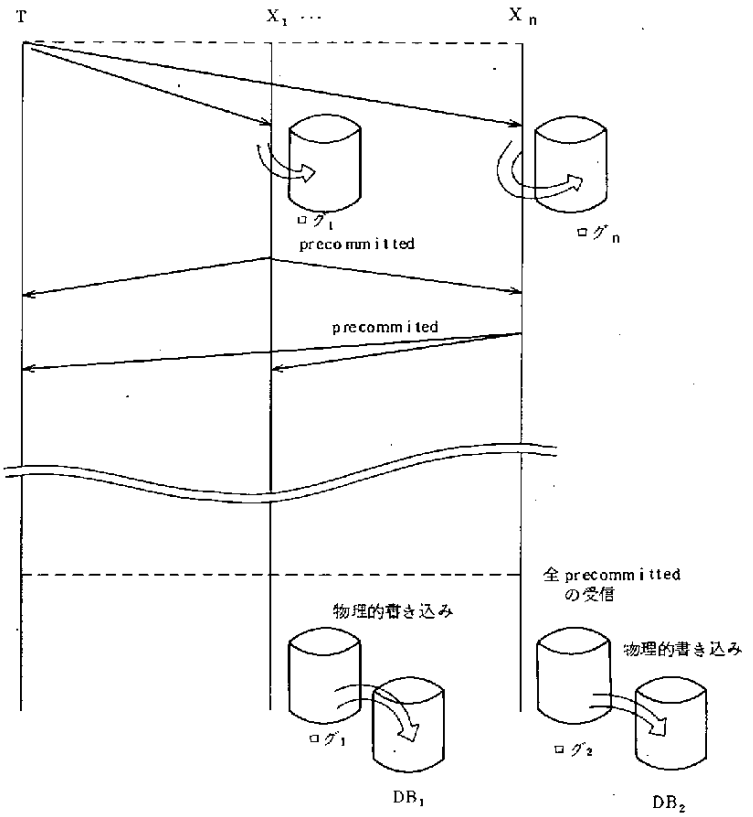


図2.20　コミットメント手順

2.3.3 CODASYL データベースシステム上の関係型モデルインタフェース（ＲＩＰ）

種々のモデル型のデータベースシステム（ＤＢＳ）と利用者間でのインタオペラブルな関係（相互運用関係）を保つ為の１つの方法は，ＤＢＳが利用者に対して共通な型のデータモデルを提供することである。共通型としては，次の理由から，関係型が望ましい。

(1) 単純さ。単純なデータ構造と，非手続的操作演算。

(2) 閉包性，即ち，検索結果を，データベース内の関係と見て，操作出来る。

既に，CODASYL 型のＤＢＳに対する非手続的な関係型インタフェースとしては，ローカルデータベースプロセッサ（ＬＤＰ）〔Takizawa８０，８２，８３，８４〕が，実働している。現在，ＬＤＰは，当協会のＭ－３６０Ｒ（富士通）のCODASYL 型のＤＢＭＳ　ＡＩＭ／ＤＢを用いたＤＢＳ上で，実働しており，以下の機能を有している。

(1) Quel と同様な形式の問合せ言語ＲＱＬの提供。

(2) ＲＱＬから，COBOL DML プログラムを生成し，これをＡＩＭ／ＤＢのＤＢＳ上で実行させる。結果をＲＤＳ（関係作業領域（ＡＩＭ／ＤＢで実現）に関係として格納する。

(3) ＲＱＬとして，group-by 付きの集約関数（例．sun，count，max，aver）の提供。

(4) セット型のメンバシップクラスに対応したインテグリティ制約を満たす更新演算の提供（ＲＱＬ）。

(5) 目標のＤＢＳの情報（スキーマ，パフォーマンス）を，ＨＩと呼ぶＤＤ／Ｄ（これも，ＡＩＭ／ＤＢで実現）に格納することにより，どのＤＢＳにも適用できる。

ＬＤＰの数年間にわたる運用を通して，以下の点が問題となっている。

(1) 関係型モデルの特徴の一つである閉包性が提供されない。

(2) 性能の問題

(3) 利用者インタフェースの問題

(1)の問題は，ＬＤＰを用いてCODASYL 型データ構造に対応した基本関係から導出された関係を，基本関係と同様に，ＲＱＬにより操作出来ないことである。これは，ＲＱＬが，CODASYL 型モデルと同等であることから，

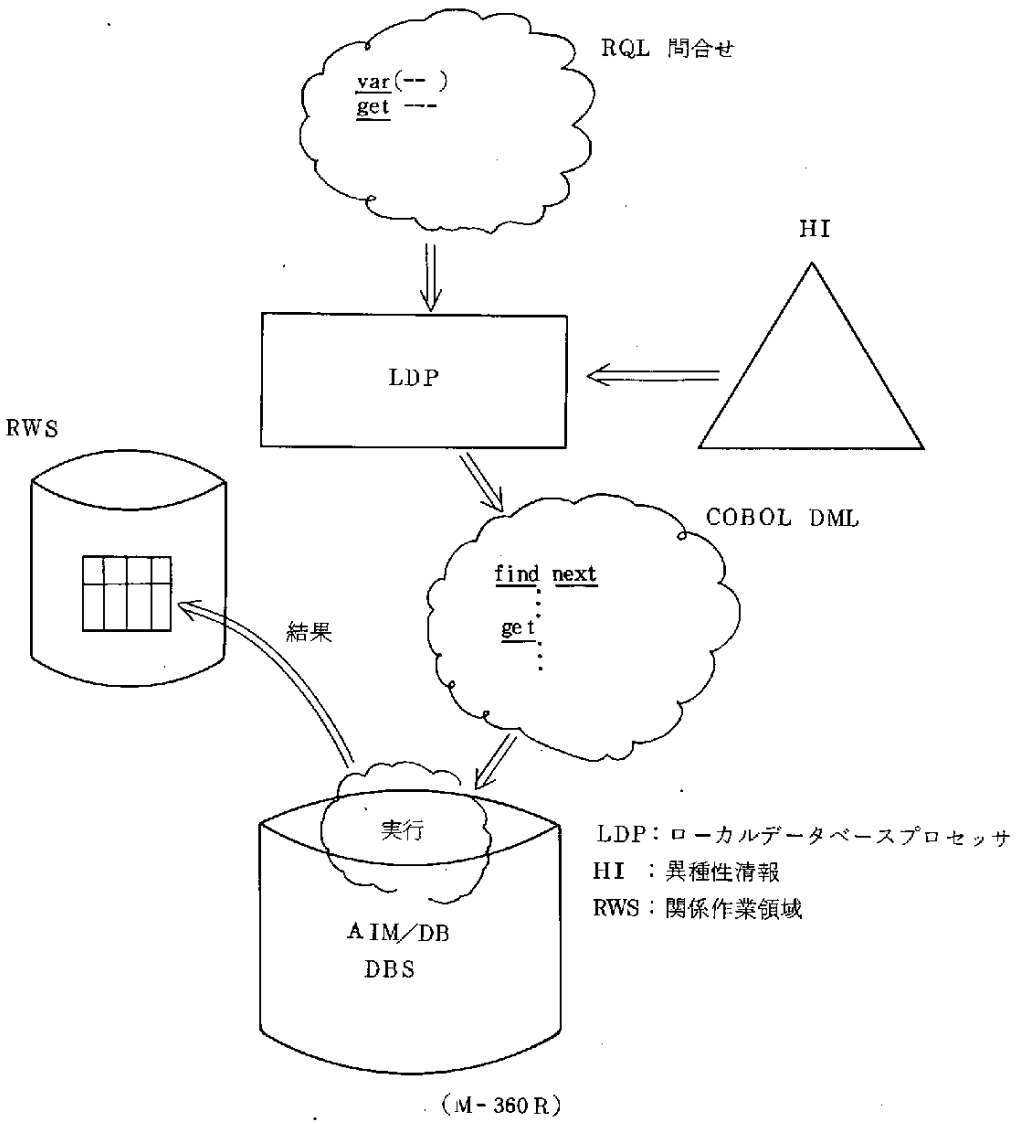


図 2.2.1　LDP

CODASYL 型モデルが閉包性を提供しない為である。

(2)は，主に，RQLより毎回COBOL DMLプログラムを生成し，1つのジョブとしてこれをプレコンパイル，コンパイル，リンクして実行させているので，（プレ）コンパイルとリンクの時間，及び，ジョブが起動されるまでの待ち時間が，経過時間のうちかなりの割合を占めていることから生じる問題である。

(3)は，RQLといった述語論理形式の問合せ言語は，一般の利用者には難かしいことによっている。特に，集約関数，更に，group-by 付きの集約関

数の習得は，一般利用者にかなり困難である。

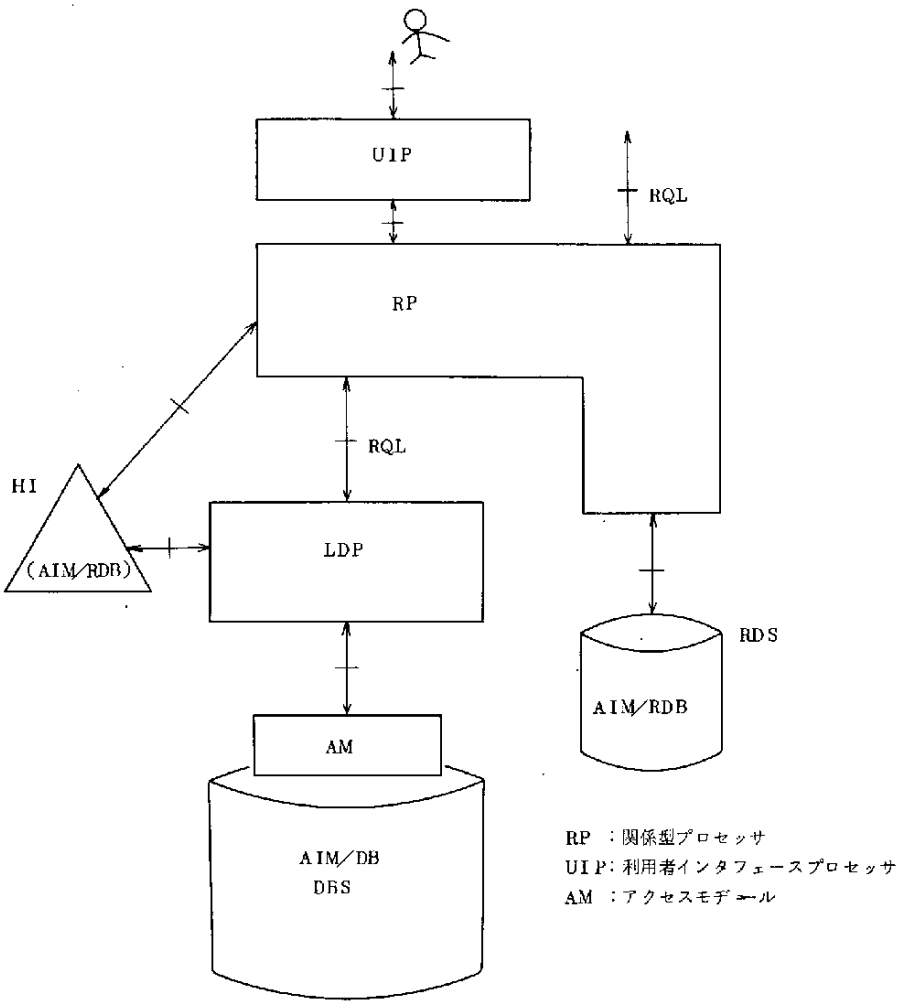


図 2.2.2　ＲＩＰ

以上の問題点を解決する為に，次の様な改良を行ったＲＩＰ（関係型モデルインタフェースプロセッサ）〔図2.2.2〕を考えている。

(1)　閉包性の提供

ＣＯＤＡＳＹＬ　ＤＢＳから導出された関係を，関係型ＤＢＳ（ＡＩＭ／ＲＤＢ）に格納する。導出された関係に対する操作は，関係型ＤＢＭＳを用いる。基本関係（即ち，ＣＯＤＡＳＹＬ型ＤＢＳ内のデータ構造）と，導出関係間

の操作は，(i)基本関係と導出関係とに対する二つの操作に分割し，(ii)前者の結果を求めて，関係型ＤＢＭＳを用いて実行する。図2.2.2のＲＰ（関係型プロセッサ）は，以上を行うモデュールである。ＲＷＳ（関係型作業領域）は，ＡＩＭ／RDB による関係型ＤＢＳである。

(2) 性能改良

プレコンパイル，コンパイル，リンクの時間と，ジョブのスケジュール待ち時間をなくす為に，アクセスモデュール（ＡＭ）を設ける。ＡＭは，各CODASYL 型ＤＢＳに固有であり，このデータ構造（スキーマ）に依存している。ＡＭは，CODASYL DBSに対する基本的な操作を行うＤＭＬプログラム（ＰＬ／Ｉ）のサブルーチンの集合で，ライブラリ化されたものである。ＬＤＰは，ＲＱＬ問合せから得られたアクセスパスに従って，ＡＭ内のサブルーチンを呼び出す。以上により，性能問題を解決出来る。

ＡＭは，データ構造依存であるが，ＨＩにCODASYL 型ＤＢＳのスキーマ情報を与えるときに，ＡＭプログラムを自動合成するようにする。これによって，ＤＢＳ毎に，ＡＭ用のプログラムを作成する必要はなくなる。

(3)については，フォーム形式の入出力，エディタイメージでの関係の更新インタフェースを設ける。

以上のＲＩＰの基本設計は終了しており，来年度以降実行予定である。CODASYL 型ＤＢＳが，ＲＩＰを設けることにより，利用者は，関係型ＤＢＳもCODASYL 型ＤＢＳも，全く同様な形式でインタオペラブルに操作出来ることになる。ＲＩＰは，現在，ＡＩＭ／ＤＢとＡＩＭ／RDB を対象としているが，他のＤＢＳへの拡張は容易である。ちなみに，ＬＤＰはＡＤＢＳ（Acos－６５０）でも実働している。

3. ＬＩＳ通信処理プロトコル

本章では，ローカル情報システム（ＬＩＳ）内の複数のプロセスの一体となった協調的な通信処理のためのプロトコル，ＬＩＳ通信処理プロトコル〔TAKIM84b,c〕について述べる。

ＬＩＳ通信処理プロトコルは，前章で述べた分散型データベースシステム（ＤＤＢＳ）の処理を効率的に行い，かつ設計を容易にすることを目的として開発したプロトコル体系である。又，群通信と呼ぶ信頼性と順序性の確保された複数プロセス間での非同期なデータ通信機能の提供を行える。

以下，群通信の概念及びＬＩＳ通信処理プロトコルの実現について述べる。

3.1 ＬＩＳ階層モデル

ＬＩＳはＤＤＢＳと，ＤＤＢＳの設計と運用とを効率的に行うための通信処理機能等を一体化したシステムである。本節ではＬＩＳの制御ソフトウェア・システムの機能，構造のモデルについて述べる。

3.1.1 階層の構成及び各階層の機能

通信網上の各ホストにおいてＬＩＳを構成するソフトウェア，ハードウェアを合せてシステムと呼ぶ。システムは副システムの集合で，階層は網内の同一レベルの副システムの集合である。〔図3.1〕

ＬＩＳの階層の構成を図3.2に示す。各階層は割り当てられた機能を行う。各階層は次の機能を持つ。

(1) 応用層

利用者固有のプロセス等が構成される。

(2) ＤＤＢ（Distributed Data Base System)層

分散型データベースシステムの管理運を行う。

(3) トランスポート群制御（ＴＣＣ）（Transport Cluster Control）層

信頼性のある群通信（後述）を行う。

(4) ＥＤＬ（Ethernet Data Link)層

本層はEthernet のデータリンクサービスのうち放送通信を行う。ＩＥＥＥのＭＡＣ（媒体アクセス制御）層に，対応しており，トークンパッシング方式でもよい。

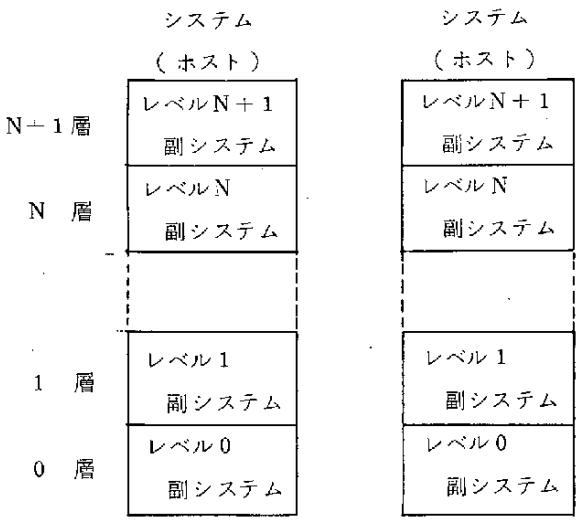


図 3.1　システム，副システム及び階層

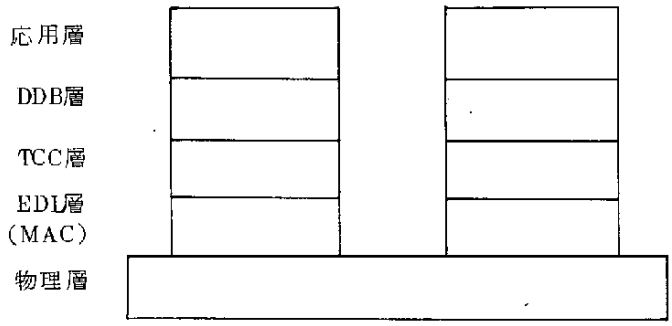


図 3.2　ＬＩＳの階層構成

(5)　物理層

物理層は伝送媒体（同軸ケーブル等）の電気的制御を行い，伝送媒体への信号の送出及び伝送媒体上の信号の受信等を行う。

副システムは実体（Entity，プロセス）の集合であり，実体はいずれかの副システム内に存在する。同一階層の実体をピアー（Peer）実体と呼ぶ。

各階層の機能は，下位層のサービスに新たな価値を付加したサービスを上位層に提供することである。サービスを行うためにピアー実体が協同する。このサービスを行うための実体間の協同に関する規定をプロトコルと呼ぶ。またサービスを上位層に提供するための階層間のインターフェースをＳＡＰ

(Service Access Point)と呼ぶ〔図3.3〕。ＳＡＰを介した上下実体間の結合には，一般に次の３つの規定がある〔図3.5〕。

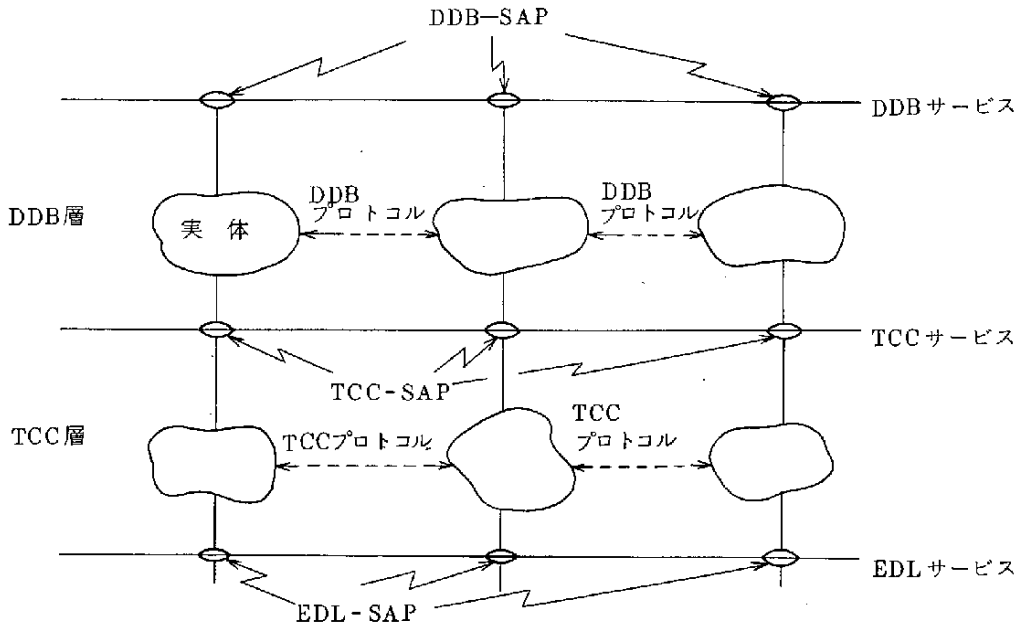


図3.3　階層，実体及びプロトコル

(i)　１つの実体は複数のＳＡＰを上位に提供し得る。

(ii)　１つのＳＡＰからは１つの上位実体のみがサービスを受けられる。

(iii)　１つの実体は複数のＳＡＰからサービスを受けられる。

3.1.2　ＬＩＳ通信処理

ＬＩＳにおける分散型データベースシステムの通信処理は，トランスポート群制御（ＴＣＣ）層で実現される単純群サービスと呼ばれるサービスを用いて行われる。以下，群通信の定義及び群通信のモデルについて述べる。

Ａ．群通信の定義

(a)　網内の通信

網内での情報の移動が通信であり，通信を行う主体が実体（E_iで表す）

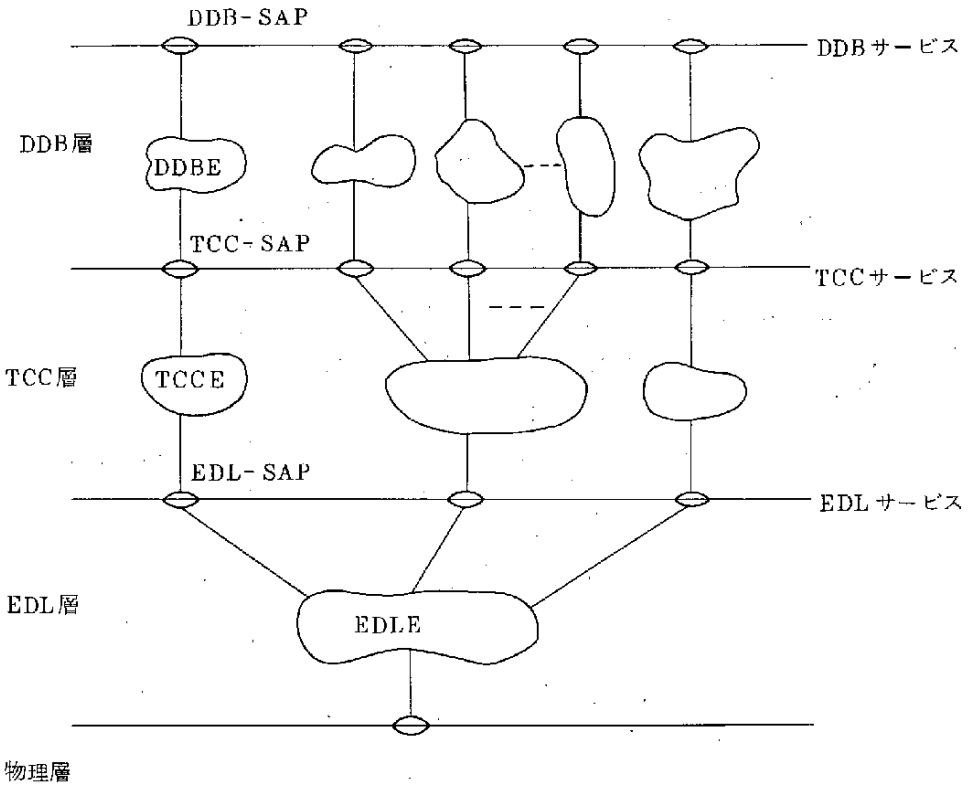


図 3.4　ＬＩＳにおける各実体の結合関係

である。1つの情報に対して情報を発する実体を送信実体，情報を受ける実体を受信実体と呼ぶ。

〔定義１〕

各実体E_i（$i=1,\cdots,\ n$）に対して

$S_i(m)$≙E_iからの情報mの送信を表す事象。

$r_j(m)$≙E_jによる情報mの受信を表す事象。

〔定義２〕

任意の事象αとβに対して次の関係を定める。

$\alpha < \beta$　$\underline{iff}$　αはβ以前に生起している。

$\alpha \equiv \beta$　$\underline{iff}$　αとβは同時に生起している。

$\alpha \leq \beta$　$\underline{iff}$　$\alpha < \beta \vee \alpha \equiv \beta$

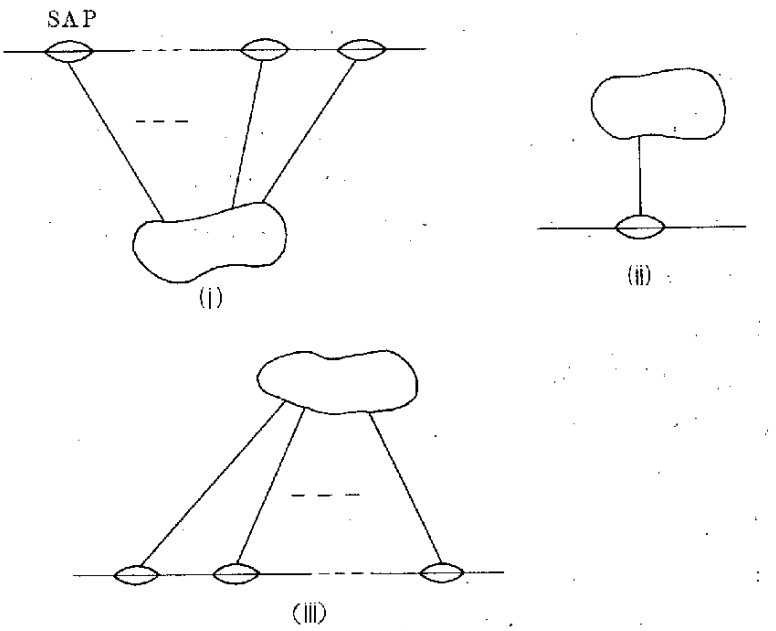


図 3.5　ＳＡＰの提供及び利用形態

〔公理〕

任意の情報 m に対して，受信事象 $r_j(m)$ が存在すれば必ず $S_i(m) < r_j(m)$ なる $S_i(m)$ が一つだけ存在する。

即ち，送信がなければ受信もなく，受信が起こるのは送信が以前に起きたときのみである。

(b)　コネクション（Connection）

〔定義３〕

二つの実体 E_i と E_j のコネクションとは，以下の性質を満たす論理的通信路である。

O 1.〔信頼－対一通信〕

$\forall S_i(m)$ に対して $(\exists\, r_j(m)\ S_i(m) < r_j(m))$

$\forall r_j(m)$ に対して $(\exists\, S_i(m)\ S_i(m) < r_j(m))$

即ち，E_i から送信された情報のみが必ず E_j で受信される。

O2.〔 F I F O性 〕

$\forall S_i(m),\ S_i(m')$に対して　$\exists / r_j(m),\ r_j(m')$,

$S_i(m) < S_i(m') \rightarrow r_j(m) < r_j(m')$。

逆に

$\forall r_j(m),\ r_j(m')$に対して　$\exists / S_i(m),\ S_j(m')$,

$r_j(m) < r_j(m') \rightarrow S_i(m) < S_i(m')$である。

即ち，E_iから送信された情報は，E_iの送信した順序に従ってE_jで受信される。

〔 仮定 〕

各実体E_iに対して，

(i)　$\forall S_i(m),\ S_i(m')$に対して

$\forall S_i(m) < S_i(m') \vee S_i(m') < S_i(m)$

即ち，同時に二つ以上の情報を送信できない。

(ii)　$\forall r_j(m),\ r_j(m')$に対して

$\forall r_j(m) < r_j(m') \vee r_j(m') < r_j(m)$

即ち，同時に二つ以上の情報を受信できない。

(C)　群の通信

〔 定義 4 〕

n（≥ 2）個の実体$E_1, \cdots, En$間の群（cluster）$C(E_1, \cdots, En)$とは，以下の性質を満す論理通信路である。

C 1〔 信頼放送性 〕

$\forall i,\ \forall S_i(m),\ \forall j(\neq i)$に対して

$\exists / r_j(m),\ S_i(m) < r_j(m)$

$\forall j,\ \forall r_j(m)$に対して　$\exists / S_i(m)$,

$\forall \ell(\neq i, j)\ \forall r_\ell(m)\ S_i(m) < r_j(m) \wedge S_i(m) < r_\ell(m)$

即ち，任意のE_iで送信された情報mは必ずE_i以外の全実体$E_j(j \neq i)$で受信される。

C 2〔 F I F O性 〕

$\forall i,\ \forall S_i(m),\ S_i(m')$ に対して $\forall j(\neq i)$ $\exists / r_j(m),\ r_j(m')$,

$S_i(m) < S_i(m') \rightarrow r_j(m) < r_j(m)$

逆に

$\forall j,\ \forall r_j(m),\ r_j(m')$ に対して $\exists / S_i(m),\ S_i(m')$

$$r_j(m) < r_j(m') \rightarrow S_i(m) < S_j(m')$$

即ち，任意の実体E_iから送信された情報は，E_i以外の全実体E_j（$j \neq i$）で送信された順に受信される。

〔定義5〕

n（≥ 2）個の実体E_1，…，En 間の群は以下の条件を満すとき単純群（simple cluster）であるという。

C1，C2

C3〔単一通信路性〕

$\forall E_i, E_j, \forall S_i(m), S_j(m')$ に対して $\forall \ell (\neq i, j), \forall k (\neq i, j, \ell)$

$$r_\ell(m) < r_\ell(m') \rightarrow r_k(m) < r_k(m')$$

B．群通信モデル

群通信はＴＣＣ層が提供するサービスで，ＬＩＳ通信処理は群通信のうち単純群通信が主体となっている。

単純群サービスは非信頼なＥＤＬ層の放送路サービスを利用して，ＴＣＣ層で信頼放送性，順序性及び単一放送路性を付加して実現される。

(a) 放送路サービス

放送路サービスとは，任意の実体の送信が同階層のある実体集合によって受信されるサービスで信頼性は有していない。

ＥＤＬ層は全ＥＤＬ－ＳＡＰの間に放送路を設定し，ＴＣＣ実体（ＴＣＣＥ）に放送路サービスを提供する。ＥＤＬ－ＳＡＰ内の放送路サービスを受ける点を放送路端点（ＣＥＰ）と呼ぶ。放送路はＬＩＳ内に唯一存在する。

(b) 単純群サービス

単純群サービスは必要なＴＣＣ－ＳＡＰ間に信頼放送性，順序性及び単一通信路性を有する論理的放送通信路（単純群）を構成する。単純群はＬＩＳ内に同時に複数存在し得る。

単純群サービスを提供するＴＣＣ－ＳＡＰ内の各点を群端点(Cluster End Point) と呼ぶ〔図3.6〕。群端点は群端点識別子によりＬＩＳ内で一意に識別し得る。また単純群は群識別子によりＬＩＳ内で一意に識別し得る。

以下，本文中で単に群という場合単純群を示すものとする。

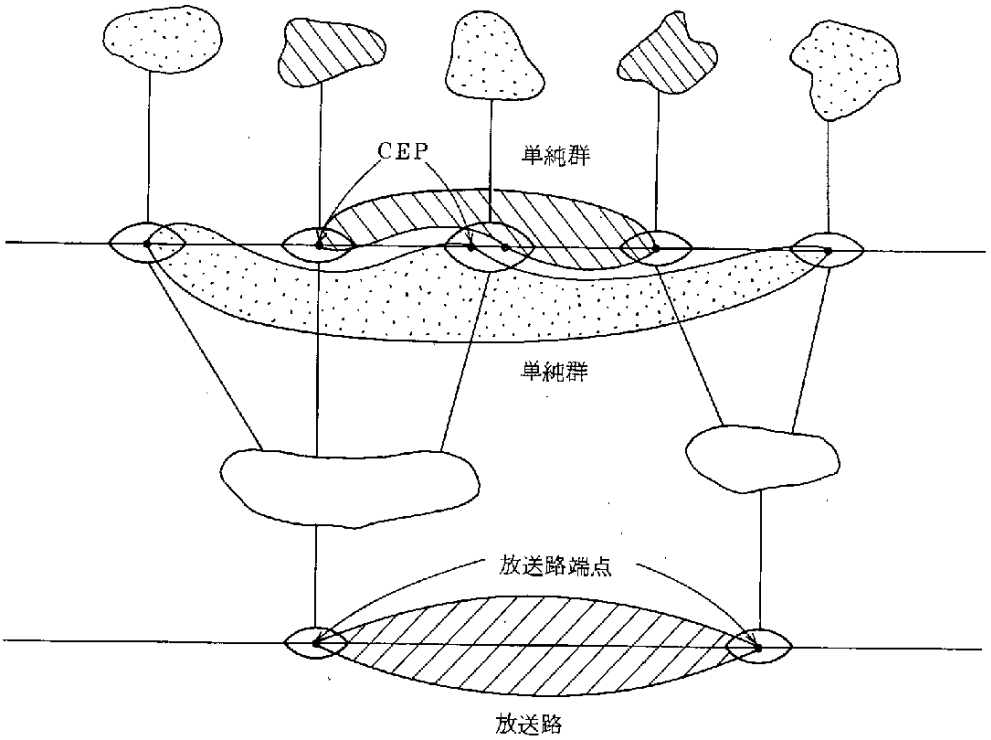


図 3. 6　放送路と単純群

(c)　群の識別

群が開設されている状態をその群のインカーネーションと呼ぶ。群の識別はインカーネーションの状態により２種類ある。１つは群の開設のための手続を行っている間の群で，開設期の群という。その識別はＴＣＣ－ＳＡＰ識別子S_i（TCC−SAPはTCC層内で一意な番地を持つ）とＣＥＰ識別子ℓ_i（ＣＥＰＩＤ）の対（ローカル群識別子，ＬＣＩＤ）の組（開設期群識別子，ＯＣＩＤ）｛（S_1, ℓ_1），…，（S_i, ℓ_i）｝で識別される。群のインカーネーションが障害等により終了した後ただちに群の再開設が行われ新たなインカーネーションが起った場合，古いインカーネーションの情報と，新しいインカーネーションの情報とを識別する必要がある。ＣＥＰＩＤとしては，このようなインカーネーションの識別が可能なように群の開設（又は再開設）が開始された時の各システムの時刻

等を用いる。

もう1つは，群の開設手続が完了した状態における群で通信中の群という。これは群識別子（ＣＩＤ）によって識別される。

(d) 実体間の通信及び階層間の通信

任意の階層，(N)層の実体（(N)－実体）間の通信は（Ｎ－１）層の通信サービスにより行われる。(N)－実体間の通信は一定フォーマットのパケット化されたデータにより行われる。この(N)－実体間の通信に用いられるパケットを(N)－プロトコルデータ単位（(N)－ＰＤＵ）と呼び，プロトコルに必要な制御情報である(N)－プロトコル制御情報（(N)－ＰＣＩ）及び実際に通信すべき(N)－ユーザーデータ（(N)－ＵＤ）から構成される。

(N)－実体と(N)－ＳＡＰで結ばれる上位層の実体（(Ｎ＋１) 実体）の間でＳＡＰを介して授受するデータを(N)－インターフェースデータ単位（(N)－ＩＤＵ）と呼び，インターフェースを制御する(N)－インターフェース制御情報（(N)－ＩＣＩ）及び実際に転送すべき(N)－インターフェースデータ（(N)－ＩＤ）から構成される〔図3.7〕。

ＤＤＢ実体（ＤＤＢＥ）間で情報Ｉを通信する場合，ＩがDDB－UDとなり，DDB－PCIと合せDDB－PDを構成する。また，DDB－PDUがTCC－IDとなり，TCC－IDUに格納されてTCC－SAPを介してＴＣＣＥに送られる。同様の操作を逐次行い，最終的に物理層のサービスにより通信が行われるが，論理的にはTCC－SAP内のＣＥＰを介してDDB－PDUが単純群により伝送される。

3.2 トランスポート群制御プロトコル（ＴＣＣＰ）

ＥＤＬサービスを用いて単純群サービスを提供するための群通信規約であるＴＣＣＰについて記述する。

3.2.1 群通信の機能及び群の操作

Ａ．群通信の機能

ＬＩＳにおいてＤＤＢＳの効率的な運営管理を行うためには，ＬＩＳ通信処理において，単純群通信が提供され，また同時に複数の単純群通信を行い得る必要がある。このためＬＩＳ通信処理に必要な基本的な機能とし

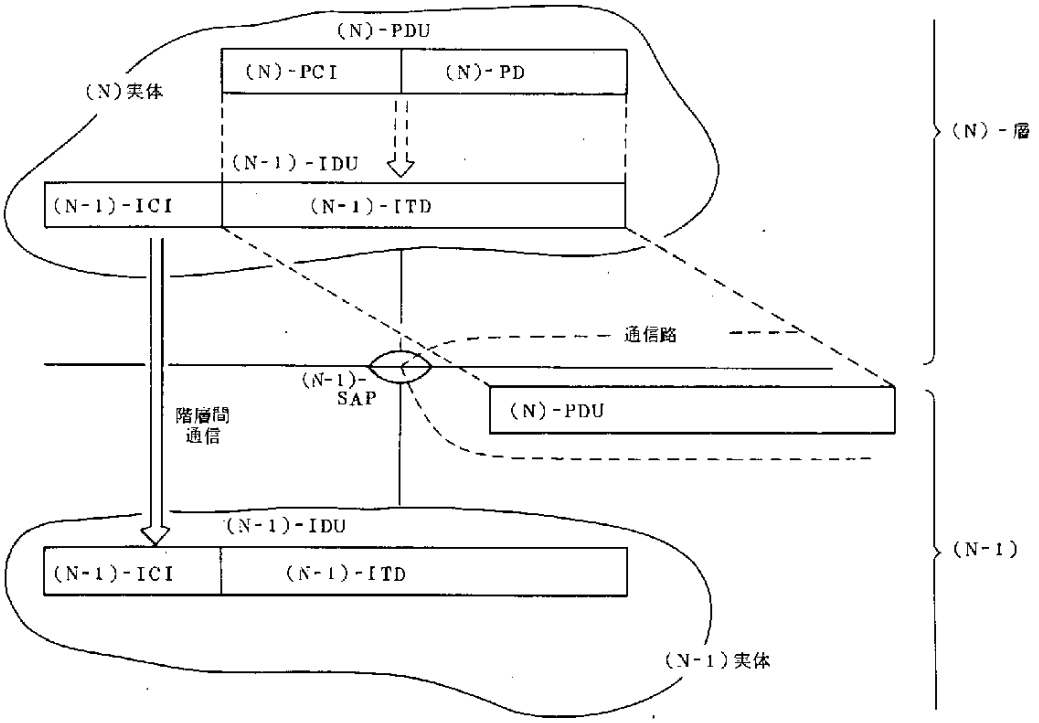


図 3.7　実体間，階層間の通信データ

しては．次の 5 つがある。

- 信頼放送性
- 順序性
- 単一放送路性
- フロー制御
- 多重化

B．群の操作

群の操作としては，

- 群の開設
- 群の解除
- データ転送
- 群への加入
- 群からの退会

の5つの操作がある。

C．群通信の段階構成

群通信は次の3段階で構成される。

(i) 開設期

(ii) 通信期

(iii) 開除期

(ii)と(iii)を合わせて，通信中という。

3.2.2 プロトコルデータ単位とインターフエースデータ単位

A．プロトコルデータ単位

(a) ＥＤＬ－プロトコルデータ単位

ＥＤＬ－プロトコルデータ単位をパケットと呼び，ＥＤＬ－ユーザーデータをそれぞれパケットヘッダ及びパケットデータと呼ぶ。パケットのフォーマットはEthernet パケットと等しい〔図3.8〕。

ＬＩＳではパケットの宛先アドレスとして放送モードを用いる。送信ＥＤＬＥから送出されたパケットは網内の全ＥＤＬＥで受信される。

(b) ＴＣＣ－プロトコルデータ単位

ＴＣＣ－プロトコルデータ単位をセグメントと呼び，ＴＣＣ－プロトコル制御情報及びＴＣＣ－ユーザーデータをそれぞれセグメントヘッダ及びセグメントデータと呼ぶ〔図3.9〕。

セグメントヘッダの各フィールドは以下の意味をもっている。

- ＥＮＯ（Entity Number）

  開設期群識別子（ＯＣＩＤ）内の，自実体のローカル群識別子（ＬＣＩＤ）の順番である。値はＥＮＯ＝0, 1,… destno－1である。

- ＣＩＤ（Cluster Identifier）

  ＣＩＤは群の識別子であり，網番地（4 byte），ホスト番地（6 byte）及びソケット番地（2 byte）から成るＴＣＣ－ＳＡＰ識別子並びにＣＥＰＩＤ（2 byte）で構成される。

  群の開設期に使用するセグメントのＣＩＤは0とする。

- ＣＴＬ（Control）

  セグメントの機能を示す制御フラグの集合〔表3.1参照〕である。

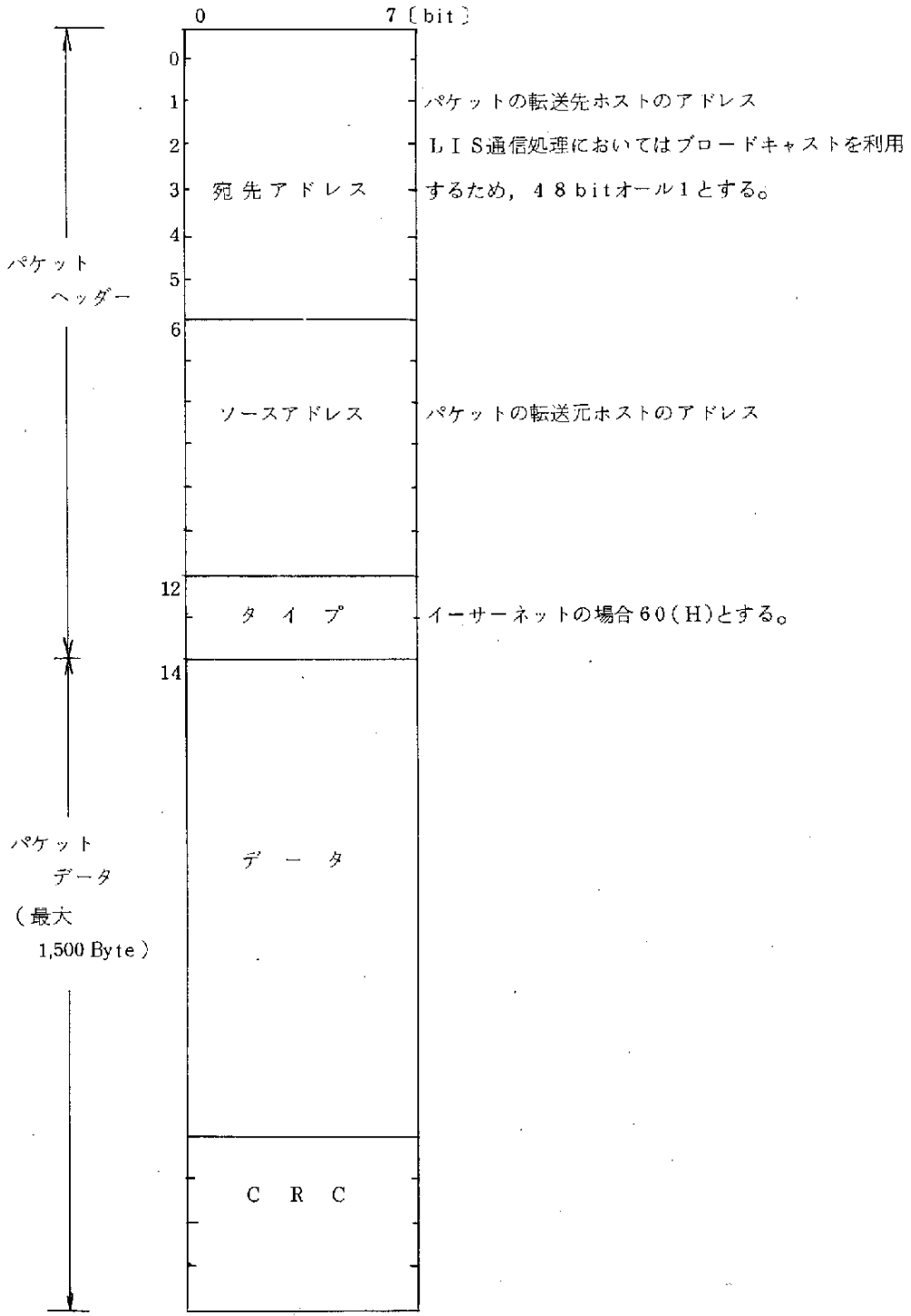


図 3.8　イーサーネットパケットフォーマット

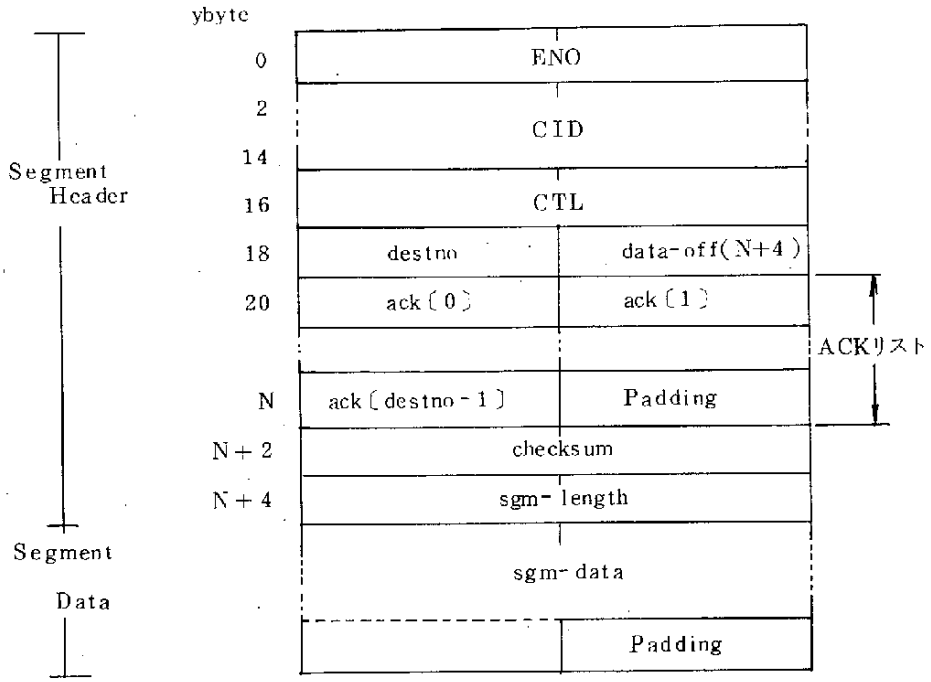
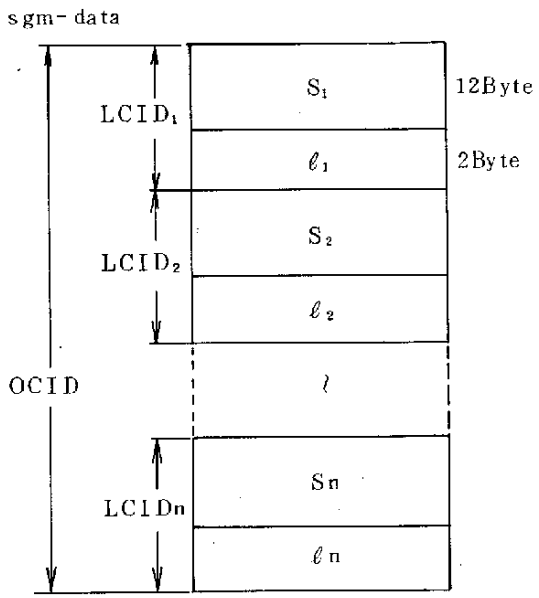


図 3.9　セグメントのフォーマット



Si＝Network address(4Byte)
＋host address(6Byte)
＋port ID(2Byte)

（注）
n＝destno
LCIDは昇順にソートされている。

図 3.10　群開設時のセグメントデータ

表 3.1　フラグテーブル

Function \ bit / Flag		0 SYN	1 FIN	2 ABORT	3	4 RST	5 RNR	6 RR	7 REJ	8 PRACK	9 ACK	A	B	C DATA	D POLL	E FINAL	F MORE	Data Field 使用有無	Data Field 内容	ACK Field 使用有無	ACK Field 内容	通常使用の有無	備考
開設期	<SYN>	○																○	OCID	×		×	
	<SYN-PAK>	○								○								○		×		×	
	<SYN-ACK>	○									○							○		×		×	
	<SYN-RST>	○				○												○		×		×	
解除期	<FIN>		○											△	△	△		△	Data	○	ACK	○	POLLとFINAL は排他使用
	<ABORT>			○														×		×		×	
通信期 RST	<RST>					○									△			×		×		×	
	<RST-PAK>					○				○					△			×		×		×	
	<RST-ACK>					○					○							×		×		×	
通信期 REJECT	<REJ>								○						△			×		○	Smallest	×	
	<REJ-PAK>								○	○					△			×		○	ACK Table	×	
	<REJ-ACK>								○		○							×		○		×	
通信期 フロー制御	<RNR>						○								△	△		×		○	Smallest	×	POLLとFINAL は排他使用
	<RNR-PAK>						○			○					△			×		○	ACK Table	×	
	<RNR-ACK>						○				○							×		○		×	
	<R R>							○							△	△		×		○	ACK	×	POLLとFINAL は排他使用
	<R R-ACK>							○			○							×		○	ACK	×	
通信期 データ転送	<DATA>													○	△	△	△	○	Data	○	ACK	○	POLLとFINAL は排他使用
	<ACK>										○					△		×		○	ACK	○	

○ destno(destination number)

群を構成するＴＣＣ実体（ＴＣＣＥ）の数である。

○ data-off(data-offset)

セグメントの先頭から，sgm-length までのバイト数。但しsgm-length は含まない。

○ ack〔i〕($i=0,\cdots,$ destno−1)(acknowledge No.)

セグメントの機能（ＣＴＬ）により内容が異なる〔表 3.1参照〕。

イ）ＣＴＬ＝＜ＦＩＮ＞or＜ＤＡＴＡ＞or＜ＡＣＫ＞の場合

各実体から受信した最後のセグメントの通番，但しack〔ENO〕はこのセグメントの通番となる。

ロ）ＣＴＬ＝＜ＲＥＪ＞or＜ＲＮＲ＞の場合

各実体から正常に受信した最後のセグメントの通番。ack〔ENO〕はこのセグメントの通番ではなく，最後に送出した＜ＤＡＴＡ＞又は＜ＡＣＫ＞の通番。

ハ）ＣＴＬ＝＜ＲＥＪ－ＰＡＫ＞or＜RNR−PAK＞の場合

Reject，Rnr処理開始後，受信した＜ＲＮＲ＞，＜ＲＥＪ＞，＜RNR−PAK＞，＜REJ−PAK＞と自分のackテーブルとの最小値。ack〔ＥＮＯ〕も同様。

ニ）ＣＴＬ＝＜REJ−ACK＞or＜RNR−ACK＞の場合

全実体で共通に認識している通番。

ホ）ＣＴＬ＝＜ＲＲ＞，＜RR−ACK＞の場合

全実体で共通に認識している通番。この前に送出した＜ＲＮＲ−ＡＣＫ＞の通番とまったく同じである。

ヘ）イ）〜ホ）以外の場合

ack〔i〕＝0

○ Padding

destno が奇数の場合に挿入する値０のバイト。

○ checksum

チェックサム　セグメント全体の語（２byte）単位の排他的論理和

○ sgm-length

sgm-data(セグメントデータ）長で，Padding部は２byte boundaryの為でsgm-lengthには含まれない。

○ sgm－data

セグメントの機能（ＣＴＬ）により，格納される内容が異なる。

イ）ＣＴＬ＝＜ＳＹＮ＞or＜ＳＹＮ－ＰＡＫ＞or＜ＳＹＮ－ＡＣＫ＞or＜ＳＹＮ－ＲＳＴ＞の場合

図3.10に示す開設期群識別子が格納される。

ロ）ＣＴＬ＝＜ＤＡＴＡ＞or＜ＦＩＮ＞の場合

上位層からのプロトコルデータ単位（ＰＤＵ）が格納される。

ハ）イ）～ロ）以外の場合

データ部なし。

(c) ＤＤＢ－プロトコルデータ単位

ＤＤＢ－プロトコルデータ単位をストリームと呼ぶ。ストリームのフォーマットについては現在未定である。

B．インターフェースデータ単位

(A) ＴＣＣ－インターフェースデータ単位

ＴＣＣ－インターフェースデータ単位をＴＣＣフレームと呼ぶ。また，ＴＣＣ－インターフェース制御情報及びＴＣＣ－インターフェースデータをそれぞれＴＣＣフレームヘッダ及びＴＣＣフレームデータと呼ぶ〔図3.11〕ＴＣＣフレームヘッダ内のコマンドの種類により，ＴＣＣフレームデータのないものがある〔表3.2参照〕。

(B) ＥＤＬ－インターフェースデータ単位

ＥＤＬ－インターフェースデータ単位をＥＤＬフレームと呼ぶ。また，ＥＤＬ－インターフェース制御情報及びＥＤＬ－インターフェースデータをそれぞれＥＤＬフレームヘッダ及びＥＤＬフレームデータと呼ぶ。ＥＤＬフレームの機能（FUNCTION CODE）によりＥＤＬフレームヘッダが異なり，ＥＤＬフレームデータのないものがある〔図3.12，表3.3〕。

C．フレームとパケットの関係

群通信においては各層のＳＡＰ間の論理通信路を用いてプロトコルデータ単位を仮想的に転送する。実際には，プロトコルデータ単位はそれぞれのフレームに格納され下位実体に転送される。下位実体に転送されると，フレームから上位のプロトコルデータ単位であるフレームデータを取り出

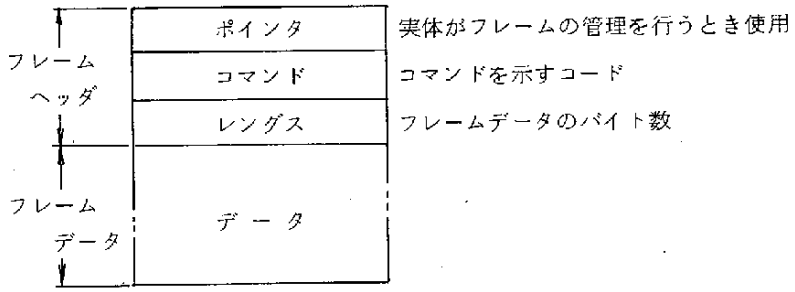


図3.11　TCCフレームフォーマット

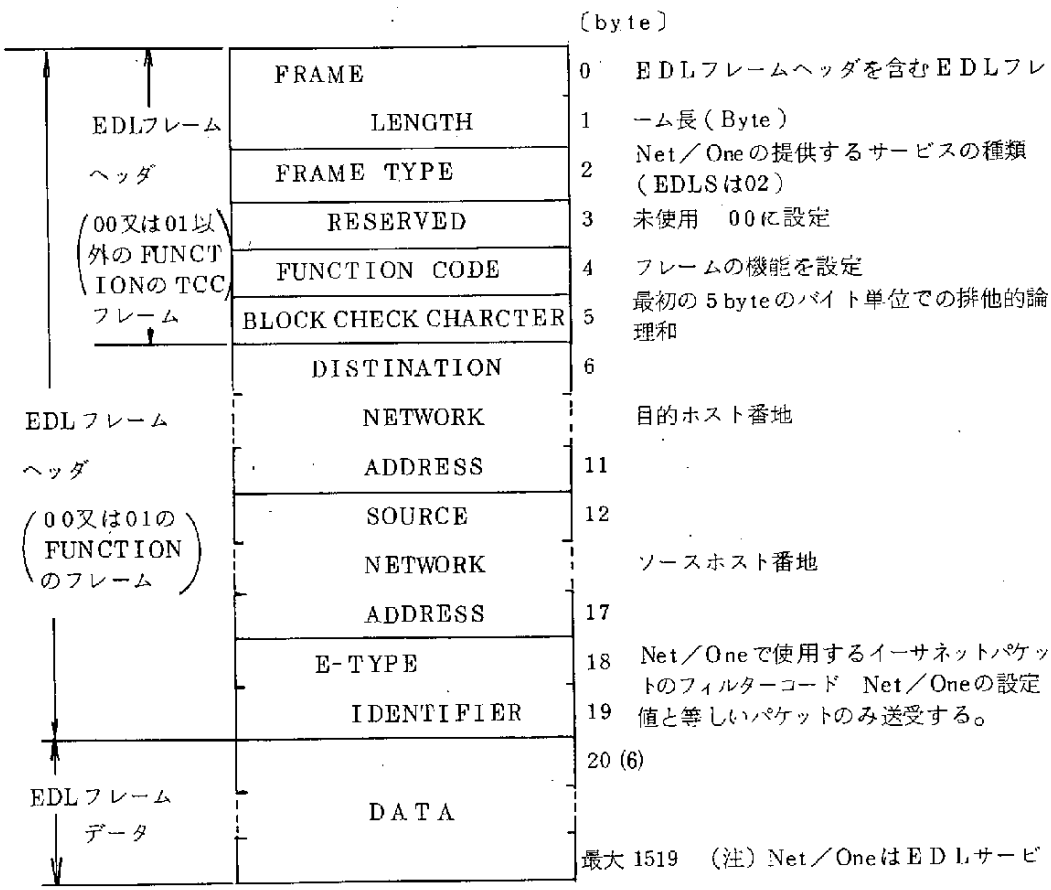


図3.12　EDLフレームフォーマット

表3.2　TCCフレーム一覧表

コマンド	コード	DDBE--TCCE	種別	フレームデータの有無	機　　能
C-ACK	2	←	管理	有	新たな群の開設要求に対する許可応答。
C-NAK	3	←	管理	無	新たな群の開設要求に対する拒否応答。
C-AOPEN	4	→	管理	有	新たな群の能動的な開設を要求する。
C-POPEN	5	→	管理	有	新たな群の受動的な開設を要求する。
C-CLOSE	6	→	管理	無	LIS 通信処理の終了要求。（正常終了）
C-CLSD	7	←	管理	無	LIS 通信処理が終了したことを示す、C-CLOSE に対する応答。（正常終了）
C-RESET	8	→	管理	有・無	各階層の初期化、または開設中の群を終了させる。
C-ABORT	9	← →	管理	有	LIS 通信処理の異常終了要求及び応答。（上位から下位は要求，下位から上位は応答）
C-STATE	10	← →	管理	有・無	LIS 通信処理の状態の確認要求及び応答。（上位から下位は要求，下位から上位は応答）
DATA	20	← →	通信	有	プロトコルデータ単位の上下実体間での転送に使用。
CLSD	23	←	通信	無	群通信の正常終了の通知。
ABORT	24	← →	通信	有	群通信の異常終了の通知。
STATE	26	← →	通信	有・無	群の状態の確認要求及び応答。（上位から下位は要求，下位から上位は応答）
ACK	27	←	通信	無	DDBEから受けたストリームの送信が完了し群内の全DDBEで受信されたことの通知。
OPND	28	←	通信	無	群の開設が完了したことをDDBEに通知。
FIN	29	← →	通信	無	群の解除を開始することの通知。
RSTD	32	←	通信	無	群通信の初期化（群が開設された時点の状態）が行われたことの通知。

し，その階層のプロトコルデータ単位に格納される。この操作を繰り返し，最終的に物理層の通信により受信ＥＤＬ実体に転送される〔図3.１３〕。

表3.3　ＥＤＬフレーム

FUNCTION CODE	TCCE ↔EDLE	機　　能
0 0	→	送信用データのフレーム
0 1	←	受信用データのフレーム
0 2	→	ＥＤＬＥが機能しているか確認する。
0 3	←	ＥＤＬＥが機能していることをＴＣＣＥに応答する。
0 4	→	ＥＤＬＥに6Byteのホスト番地の通知を要求する。
0 5	←	ＴＣＣＥにホスト番地を応答する。
1 6	→	ＥＤＬＥに対し特定の値のＥ-TYPEを設定する。
2 6	→	ＥＤＬＥに対し　特定の値以外のE-TYPEを設定する。
0 7	←	ＴＣＣＥに対し，E-TYPEの設定終了を応答する。
0 8	→	ＴＣＣＥが通信データの受信を行う。
1 8	→	ＴＣＣＥが特定の番地でのみ通信データを受信する。
3 8	→	ＴＣＣＥが特定の，番地及び放送による通信データのみを受信する。
7 8	→	ＴＣＣＥが特定の，番地，放送及び，ベンダーの放送による通信データのみを受信する。
B 8	→	ＴＣＣＥが特定の，番地，放送及び同報による通信データのみを受信する。
F 8	→	全通信データを受信する。
0 9	←	ＴＣＣＥに対し上記受信条件の設定を応答する。
0 A	←	ＥＤＬＥがリセットされたことを通知
0 B	→	ＥＤＬＥがリセットされたことの通知に対する応答
F E	→	ＴＣＣＥがＥＤＬＥのリセットを要求
F F	←	ＥＤＬＥがリセット要求に対しリセットしたことを応答

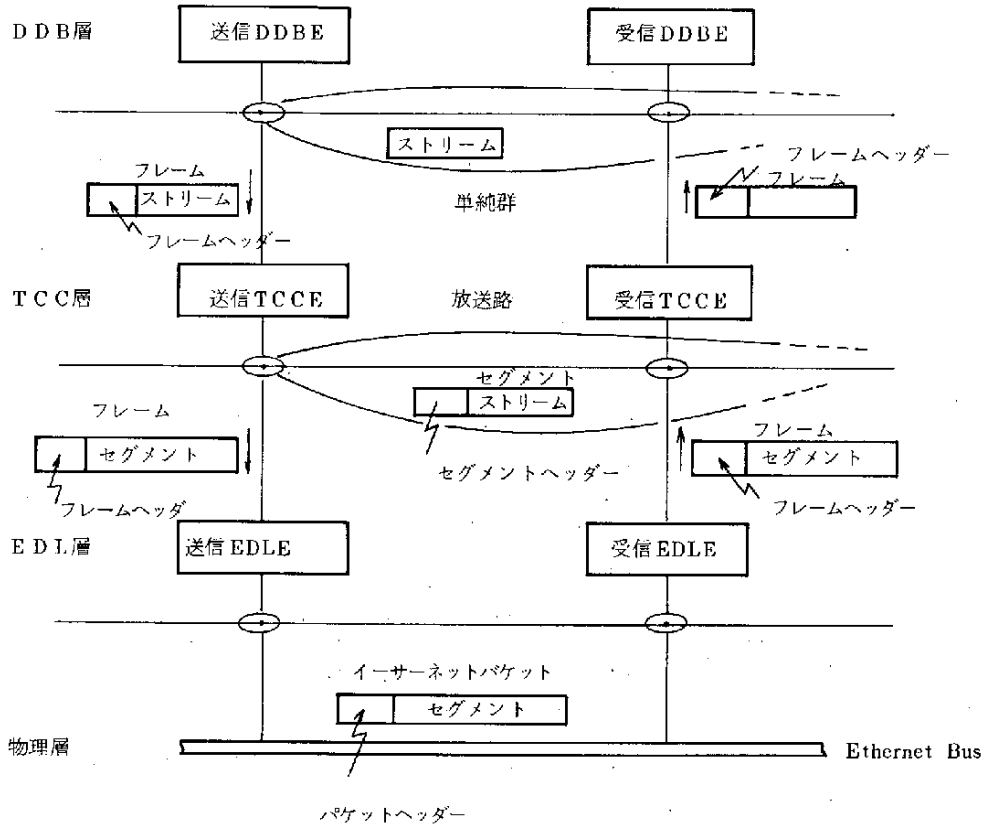


図3.13　フレームとパケット

3.2.3　群の開設及び識別

群の開設はＤＤＢ実体（ＤＤＢＥ）からの開設要求により行われる。ＤＤＢＥからの開設要求はTCC－SAPを介して行われ，群サービスはＴＣＣ－ＳＡＰ内のＣＥＰを介して提供される。また，群サービスは複数のＴＣＣ実体（ＴＣＣＥ）の協同により提供される〔図3.13〕。以下，本節の記述において単に実体という場合は群を構成するＴＣＣ実体を指す。また，群はｎ個のＴＣＣ実体で構成されるものとする。

群サービスを提供するために協同する実体の選定は開設要求に基づいて行われる。

群の開設とは，ＤＤＢＥの開設要求に基づき協同する実体の集合を定め，これら実体間の同期を取り，各実体と群サービスを受けるそれぞれのＤＤＢＥの間のＣＥＰの設定及び，同期の確立を行うことである。これらの手続を行う期間を開設期と呼ぶ。

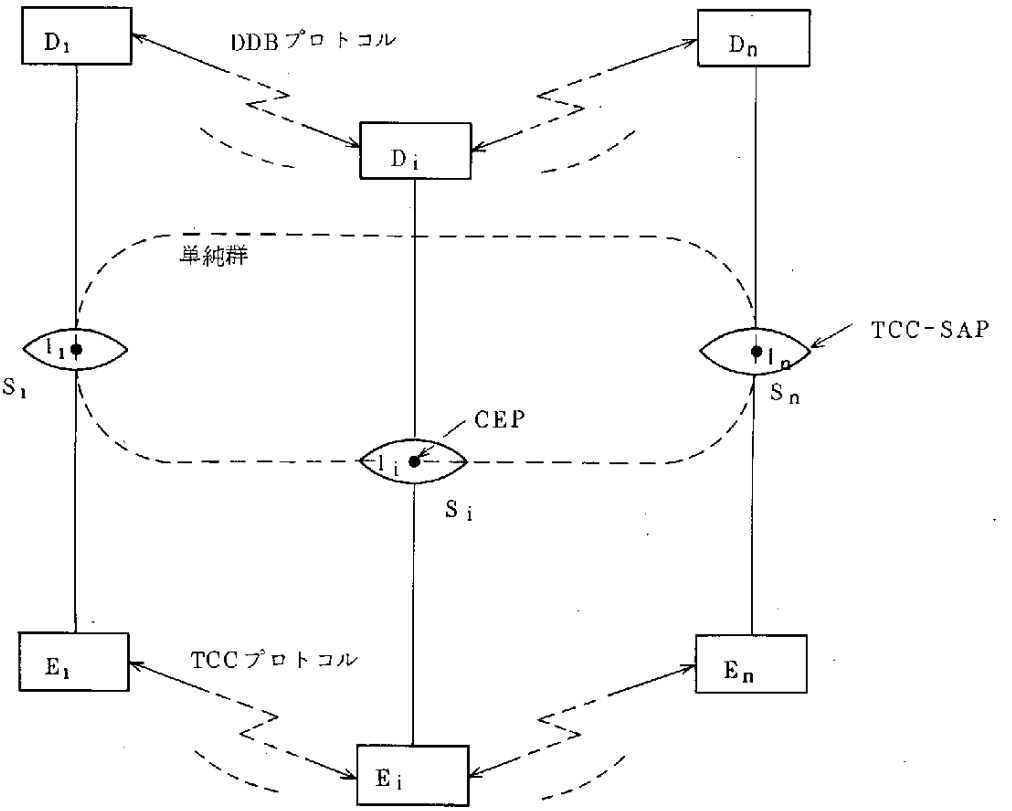


図3.14　n個のＤＤＢ実体に対する群サービス

Ａ．開設要求

群の開設要求には能動的開設要求（ＡＯＰＥＮ）と受動的開設要求（ＰＯＰＥＮ）の２種類がある。

ＴＣＣ－ＳＡＰをソケットと呼び，ＴＣＣ－ＳＡＰ識別子S_iをソケット番地とする。S_iはEthernetの網番地4 Byte 及びホスト番地6 Byte並びにソケット識別子2 Byteの計12Byte で構成される。ＡＯＰＥＮにおいては群を構

成するソケット番地を指定する。一方，POPENを行うＤＤＢＥは一般にサーバとしての役割があり，ソケット番地を指定する記名POPENの他にソケット番地を指定しない無記名POPENがある。ソケット番地の指定は自分が群サービスを受けるソケット番地を含むソケット番地の組（S_1,…, Sn）で指定する。

群の開設はある実体からのAOPENが引き金となって開始される。引金となるAOPENのソケット番地の組を$（$S_1$,… Sn）としたとき次の条件が満されない場合群の開設手続は中止される。

〔条件〕

ソケット番地の組$に示されたソケットを提供するＴＣＣ実体が

Ａ－１　ソケット番地の組$でAOPENされる。

Ａ－２　ソケット番地の組$で記名POPENされる。

Ａ－３　無記名POPENされる。

上記３種類のいずれかで，すべて開設要求を受ける。

Ｂ．開設期の群の識別

開設期の群の識別は開設期群識別子（OCID）で行う。ＯＣＩＤはソケット番地Siとソケット内のＣＥＰ識別子（CEPID）ℓiの対の組$O=\{(S_1, \ell_1), \cdots, (Si, \ell i), \cdots, (Sn, \ell n)\}$で表される。$Si$は，Ethernet の網番地4 Byte及びホスト番地6 Byte並びにソケット識別子2 Byteの計12 Byteで構成される。ℓiは2 Byteで各システムのローカル時刻を用いる。ℓi の値は$0 < \ell i \leq 2^{16}-1$である。また開設期群識別子O内で$\ell i = 0$はℓiに関する情報がないことを示し，$O=\{(S_1, \ell_1), (S_2, 0), \cdots, (Sn, 0)\}$を$O=\{(S_1, \ell_1), S_2, \cdots, Si, \cdots, Sn)\}$のように0を略して記す。あるインカーネーションにおいて，ℓiが$\ell i \neq 0$として２つ以上の値を持つことはなく，値が異なる場合はインカーネーションが異なることを示す。

Ｃ．群の開設

ｎ個のＤＤＢ実体$D_1, \cdots, Di, \cdots, Dn$に対して，ｎ個のＴＣＣ実体$E_1, \cdots, Ei, \cdots, En$が，各々対（$Di$–$Ei$）となって，協同して群サービスを提供する〔図3.14〕。このため，群の構成員（$E_1, \cdots, Ei, \cdots, En$）間で協同のための同期を確立する必要がある。同期は全構成員の群端点識別子（CEPID）

を認識し，全構成員が共通の開設期群識別子（ＯＣＩＤ）を認識することにより確立される。共通のＯＣＩＤの認識とは，以下の条件を満足していることである。

群の構成員を$E_1, \cdots, E_i, \cdots, E_n$としたとき，

Ｂ－１　各E_iが他の全E_jのソケット番地S_jとCEPIDℓ_jの対(S_j, ℓ_j)を知っている。

Ｂ－２　各E_iは，他の全E_jが自分のℓ_iを知っていることが判っている。

Ｂ－３　各E_iは他の全E_jがⅰ)，ⅱ)の状態であることを知っている。

また，同期を確立するために構成員間でセグメントを用いてＯＣＩＤを送受する。この同期を確立するための手続きを開設手順と呼ぶ。開設手順はＢ－１からＢ－３の３つの条件を満すため，DARPAのＴＣＰの３方向ハンドシェークプロトコル〔DARPA８１〕拡張した３方向放送ハンドシェーク（Three－way Broadcasting Handshake，TBH）手順を用いる〔図3.15，図３.１６〕。開設手順で使用するセグメントは，ＳＹＮ，ＳＹＮ－ＰＡＫ，ＳＹＮ－ＡＣＫ及びＳＹＮ－ＲＳＴの４種類で，セグメントヘッダ内のＣＴＬフィールドのＳＹＮフラグがいずれもＯＮとなっている。また，ＳＹＮ－ＰＡＫ，ＳＹＮ－ＡＣＫ及びＳＹＮ－ＲＳＴは，それぞれCTLのＰＡＫ，ＡＣＫ及びＲＳＴがＯＮとなっている。セグメントの機能はこのようにフラグの組み合せで指定される〔表３.１参照〕。以下の各セグメントは，セグメントヘッグ内のＣＩＤは０であり，セグメントデータ内にＯＣＩＤが格納される。セグメント内に格納されたＯＣＩＤを表わすためＳＹＮ(O)(O＝$\{(S_1, \ell_1), \cdots, (Si, \ell i), \cdots, (Sn, \ell n)\}$)のように記す。

群の開設においては，少なくとも１個以上のＤＤＢ実体D_iが，ソケット番地の組$\$=(S_1, \cdots, Si, \cdots, Sn)$を用いて能動的開設要求AOPEN($)を各E_iに対して行う必要がある。また，残りのＤＤＢ実体は，無記名受動的開設問求POPEN又は，＄で記名受動的開設要求POPEN($)を各ＴＣＣ実体に行う必要がある。開設要求を受けたＴＣＣ実体$E_1, \cdots, En$は開設手順に従い群を開設する。以下，開設手順につき能動側，受動側に分けて記述する。又，図３.１６には，開設手順を表す状態遷移図を示す。

〔能動側〕

ⅰ）能動的開設要求AOPEN($)($=$(S_1, \cdots, Si, \cdots, Sn)$をDiから

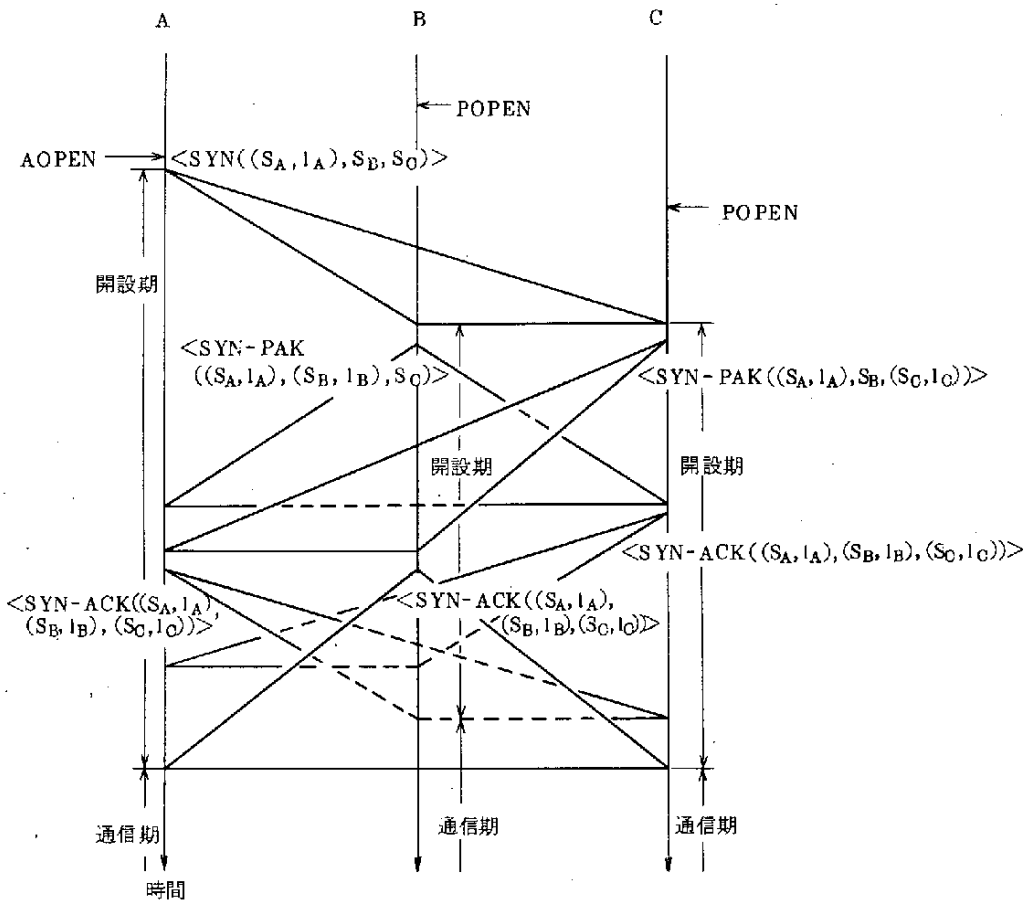


図3.15　群開設例

受けたEiは，SYN(S1, …, (Si, ℓi), …, Sn)を放送し，状態SYN−SENTに移行し，全実体からSYN(O)(O={(S1, ℓ1), …, (Si, ℓi), …, (Sn, ℓn)})又はSYN−PAK(O)を受信するまで待つ。

ii) 全実体からSYN(O)又はSYN−PAK(O)を受けた時点でEiは条件B−1，B−2を満足する。この時点でSYN−ACK(O)(全ℓi≒0)を放送し，PRE−ESTABLISHEDの状態に移る。但し，群を構成する実体が2個で，他方の実体がPOPENの場合はESTABLISHEDの状態に移る〔図3.17(2)〕。

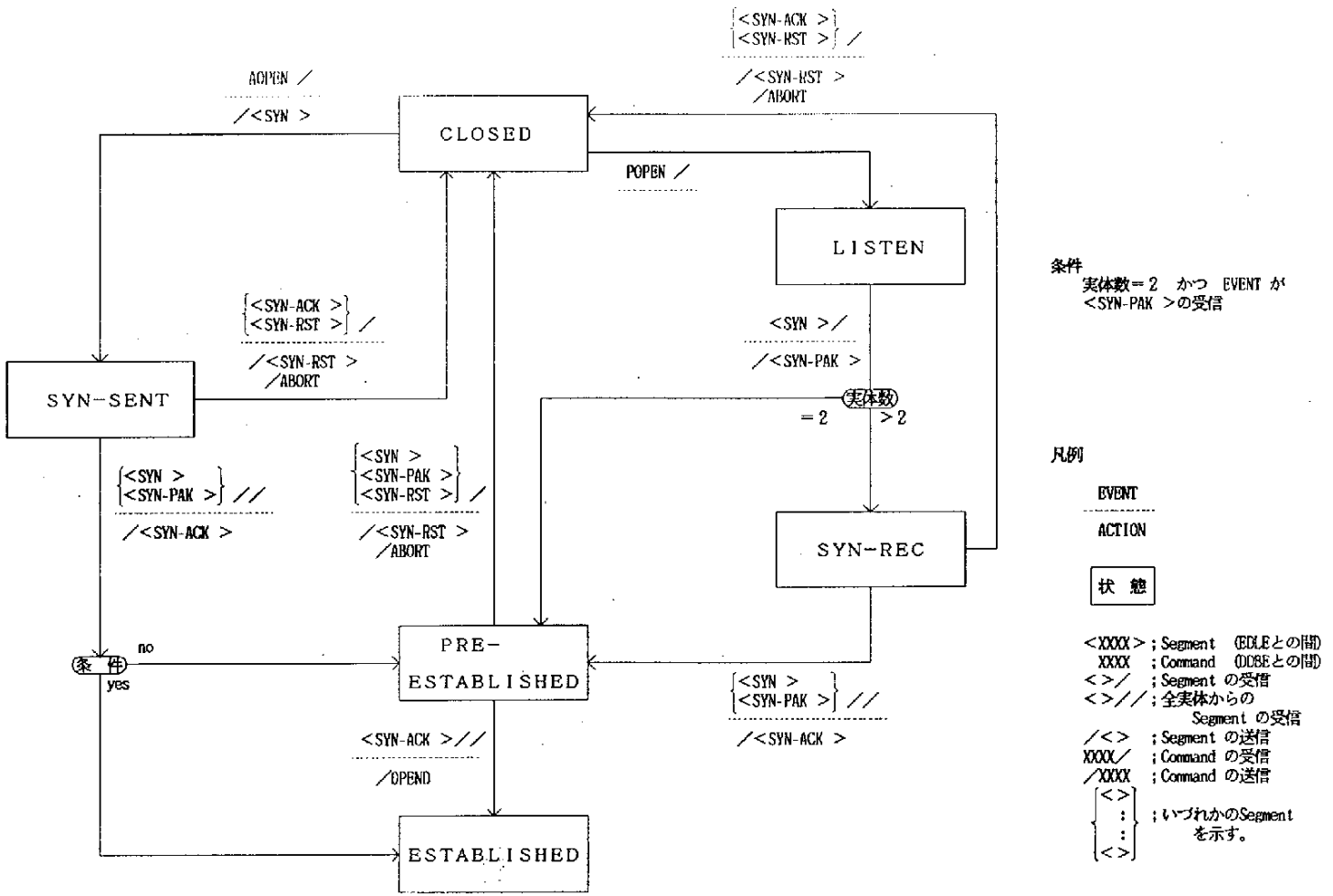


図3.16　開設手順の状態遷移図

iii) E_iは実体からSYN−ACK(O)(全$\ell i \neq 0$)を受信すると条件B−3を満足し通信期(ESTABLISHEDの状態)に入る。

iv) SYN−SENTの状態でE_iがSYN−ACK(O)を受信するということは，E_iがSYN−PAK(O)を1個以上受信し得なかったことを示し，群の開設は中止される。また，PRE−ESTABLISHEDの状態でSYN(O)又はSYN−PAK(O)を受信した場合は，他のいずれかの実体がPRE−ESTABLISHEDの状態に移行し得ないと判定し，開設を中止する。SYN−RST(O)は開設の中止を指示するセグメントでSYN−RST(O)を受信した実体はSYN−RST(O)を放送し開設を中止する。

〔受動側〕

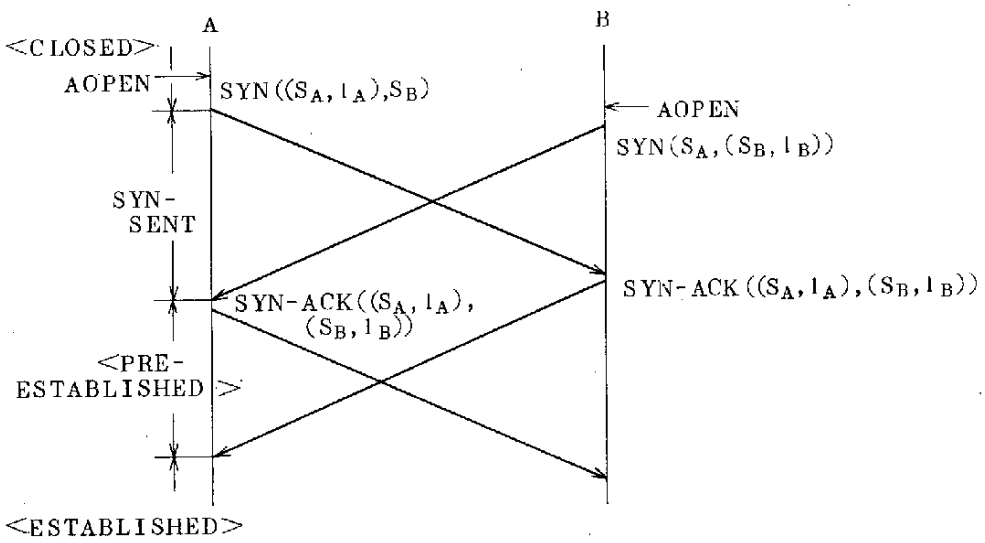
i) 受動的開設要求POPEN($)($=(S_1, …, Si, …, Sn))又は無記名のPOPENをD_jから受けたE_jはSYN(S_1, …, (Si, ℓi), …, Sn)の受信を待つ(LISTEN状態)。SYN(S_1, …, (Si, ℓi), …, Sn)を受信すると自身のCEPIDのℓjを他実体に通知するためSYN(S_1, …, (Sj, ℓj), …, Sn)を放送し，SYN−RECの状態に移行する。但し，群を構成する実体が2個の場合はPRE−ESTABLISHEDの状態に移行する〔図3.16(2)〕。

ii) SYN−RECの状態で全実体からSYN(O)(O＝{(S_1, ℓ_1), …, (Si, ℓi), …, (Sn, ℓn)})又はSYN−DAK(O)を受けた時点でE_jは条件B−1を満す。この時点でSYN−ACK(O)(全$\ell i \neq 0$)を放送し，PRE−ESTABLISHEDの状態に移行する。

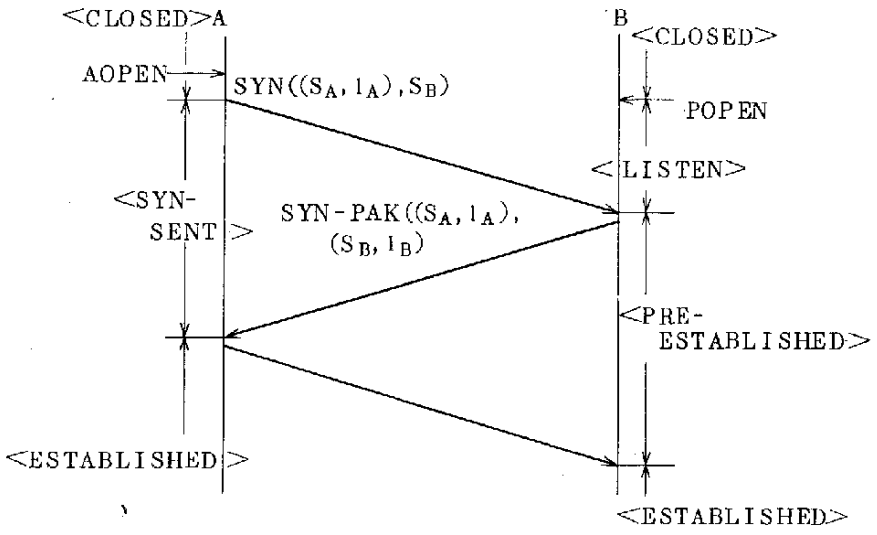
iii) E_jは全実体からSYN−ACK(O)(全$\ell i \neq 0$)を受信すると条件B−2，B−3を満足したこととなり，通信期(ESTABLISHEDの状態)に入る。

iv) SYN−RECの状態でE_jがSYN−ACK(O)を受信するということは，E_jがSYN(O)又はSYN−PAK(O)を1個以上受信し得なかったと判断し開設を中止する。

開設中の開設期群識別子(OCID)をO_1としたとき，O_2は$O_2 \neq O_1$なるOCIDとする。このときO_2を持つセグメントを受信したE_iは，O_2



(1) AOPENを受けた2実体による群の開設



(2) AOPEN及びPOPENを受けた2実体による群の開設

図3.17 2実体による群の開設

を古いインカーネーションと判定しSYN−RST(O_2')を放送する。O_2'は以下の手順で求められる。

$O_p=((S_1, \ell_1^p), \cdots, (Si, \ell i^p), \cdots, (Sn, \ell n^p))$ $(p=1 \text{ or } 2)$

としたとき

i) $\forall i$に対して$\ell^1 i, \ell^2 i \neq 0$かつ$\ell^1 i \neq \ell^2 i$のとき

$O_2'=O_2$

ii) $\forall i$に対して$\ell^1 i, \ell^2 i \neq 0$　$\exists i$　$\ell^1 i=\ell^2 i$のとき

$O_2'=O_1$

SYN−RST(O_2')により古いインカーネーションと新しいインカーネーションの混在も防止することができる。

OCIDOはセグメント内のセグメントデータ部に格納される。これは，n個の実体間で群を構成するのにn×14 Byteが必要で，セグメントヘッダが大きくなるのを防止するためである。群の開設が完了した段階で，O内ローカル群識別子（LCID）($Si, \ell i$)を14 byteの値として，この最小値を群識別子（CID）に採用する。以降，群の識別はCIDによる。また，各実体は，($Si, \ell i$)の値により昇順に実体番号（ENO＝0, 1,…，n−1）が与えられる。

3.2.4　データ転送

群の開設が完了するとDDB実体は各SAPを介して，単純群サービスを受け，全実体間で非同期にデータ転送を行うことができる。各DDB実体Diは送受信データをストリーム単位で送受する。群サービスを提供するTCC実体EiはDiからデータフレームによりストリームを受ける。ストリームは可変長であり，Eiはセグメントに格納し得るようにストリームを分割する。また，各実体Ejはセグメントを受信すると送信実体EiがDiから受けたストリームと等しい形にセグメント内のデータを編集する。ストリームの区切りを表すためセグメントのコントロールフラグ（CTL）のMOREビットを用いる。MOREビットが1のセグメントは同一ストリームのデータが続くことを意味し，MOREが0のセグメントは，セグメント内のデータが一連のストリームの最後のデータであることを意味する。

以下，単純群サービスを提供するための信頼放送性，順序性，単一放送路性，非同期データ通信の実現方法及びフロー制御，エラー制御等の補助的機

能の実現方法について記述する。また群の解除手順について記述する。

A．単一放送路性

ＬＡＮを用いた単一放送路性の実現は容易である。同軸を用いたＣＳＭＡ／ＣＤ(Carrier Sensed Multiple Access with Collision Detection)方式〔METCR76〕の場合，ある時点に同軸上で行える通信は唯一である。また，トークンパッシング方式においても，トークンは唯一存在するのみである。このように，ＬＡＮでは送信を行える実体は，ある時点で１個しか存在しない。この結果，各ＥＤＬ実体が同軸上にパケットを送出した順に全ＥＤＬ実体が受信するため，障害が発生しない限り単一放送路性は実現される。また，欠落セグメント，受信不能状態等の障害発生時においても，後述の再送手順（ＲＥＪ手順）及びフロー制御手順（ＲＮＲ手順）により，単一放送路性が確保される。

B．信頼放送性及び順序性

信頼放送性の定義〔定義４，Ｃ１〕を実際の群通信に適用すると以下の条件により表される。

群の構成員（ＴＣＣＥ）を$E_1, \cdots, E_i, \cdots, E_n$としたとき，

Ｃ－１　送信実体E_iが送信したセグメントg_iは他の全実体E_jにより正しく受信される。

Ｃ－２　送信実体E_iはE_i以外の全実体E_jがg_jを正しく受信したことを知っている。

Ｃ－３　送信実体E_i以外の全実体E_jがg_iを受信したことを，他の全実体が知っている。

これらの３条件は開設時のローカル群識別子（ＬＣＩＤ）の認識に関する条件Ｂ－1,2,3と同等である。Ｃ－１の条件及びＣ－２の条件はセグメント内のチェックサム及び通番による受信報告により実現される。また，条件Ｃ－３は受信セグメントのキで操作により実現される。一方，順序性についてはセグメントの欠落，重複の防止により実現でき，Ｂ－1,2.の条件が満足されれば順序性も満される。

チェックサムはセグメント内のデータの誤り検出を行い，誤りがあった場合はセグメントの欠落と同等に取り扱うことができれ（後述）。

(A) 通番による受信確認

転送されるセグメントに通番を付ける。通番は各ＴＣＣ実体ごとに1から２５５までの値を順次サイクリックに使用する。また，0は通番を付けないセグメントに使用される〔表3.1〕。ＴＣＣ実体E_iから送信されるセグメントのS_iはセグメント内のAcknowledge(ＡＣＫ)リストＡＣＫ〔$i-1$〕(i＝ＥＮＯ)に格能される。またACK〔$j-1$〕($j \neq i$)に格納される値a_jは，E_iがE_jから最後に受信したセグメントの通番で，E_iが他の実体に行う受信報告である。実体E_iが送信するセグメントのＡＣＫリストを$G_i=\{a_1, \cdots, a_i, \cdots, a_n\}$(但し，$a_i=S_i$また，$G_i(j)=a_j$と記す)で表す。各実体$E_j$は信頼性及び順序性を確保するため，各実体$E_i$から送られる通番及び受信報告を管理するための２次元のＡＣＫテーブル$\mathbb{U}_j$を持つ。実体E_jのＡＣＫテーブルの各要素$\mathbb{U}_j(p, g)$は次の意味を持つ。

$\mathbb{U}_j(j, q)$＝実体E_qから最後に受信したセグメントの通番。但し，$j \neq q$

$\mathbb{U}_j(j, j)$＝実体E_jが最後に送信したセグメントの通番。

ＡＣＫリストにより，各実体がセグメントを受信したことの受信報告が行われる。ある実体E_iにおいて送信すべきデータがない場合，E_iからの受信報告が行われなくなる。このため，実体E_iはＡＣＫセグメントを必要により送信する。ＡＣＫセグメントにはデータ部がなく，セグメントヘッダ内のコントロール(ＣＴＬ)フィールドはＡＣＫフラグのみＯＮとなっている。また，ある送信セグメントに対し，ただちに応答がほしい場合にはＣＴＬフィールドのＰＯＬＬビットを1とする。ＰＯＬＬはデータ通信以外の制御手順においても使用され，一般に応答はＦＩＮＡＬを1としたセグメントで行う。

セグメントの欠落及び重複の検出は，次の判定により行う。

受信実体E_jが，送信実体E_iからセグメントg_iで受信したとき

Ｄ－1　g_iのＡＣＫリストG_i内のa_iと，E_jのＡＣＫテーブル$\mathbb{U}_j$内のα_{ii}(＝$\mathbb{U}_j(i, i)$)を比較し，$a_i+1=\alpha_{ii}$(mod 255)であることを判定する。

Ｄ－1－1　$a_i+1>\alpha_{ii}$の場合，g_iは重複セグメントである。

Ｄ－1－2　$a_i+1<\alpha_{ii}$の場合，g_iの受信前に欠落したセグメント

が存在する。

D－2　Gi内の akと Uj内の α_{jk}を $k \neq i$ であるすべての k について比較する。

この結果，$a_k > \alpha_{jk}$である a_kが存在する場合，Ekからのセグメントが欠落している。

この判定により重複セグメントは無視しＡＣＫのみを返す。また，欠落セグメントは再送要求（ＲＥＪ）を行う。再送要求は単一放送路性を考慮しＲＥＪ手順により行われる（後述）。また，$(a_i - \alpha_{ii}) > W$（Wはフロー制御用の定数）である場合，単純群の機能を保持した状態での再送が不可能となり，群の初期化（ＲＳＴ手順）が行われる。Ｄ－1,2の条件が満された場合，$U_j(i, q)$ $(q=1,\cdots, n)$をGiで置き換え，$U_j(p, i)$ $(p=1,\cdots, j-1, j+1,\cdots, n)$のうち最小の $U_j(p, i)$と $U_j(j, i)$ を比較し，$U_j(p, i) > U_j(j, i)$であれば，$U_j(p, i)$を $U_j(j, i)$ に格納する。

次にEjがセグメントを送信する際 $U_j(j, j)$（$U_j(j, j)$はEjが最後に送信したセグメントの通番Sj）を１増加し，$U_j(j, q)$ $(q=1,\cdots, n)$をGjとしてセグメントgjのＡＣＫリストを作成する。Eiは gjを受信することにより，giのEjによる受信を確認し得る。Eiは，Ei 以外の全実体Ejからgjを受けることによりＣ－１の条件を満すことができる。

(B)　送受信セグメントのキュー操作

送信したセグメントの全実体での受信の確認（条件Ｃ－２）及び同セグメントの全実体での受信を全実体が確認（条件Ｃ－３）を，効率的に行うためにキューによる送受信セグメントの管理を行う。送信実体Eiにおいて送信セグメントgiが条件Ｃ－２を満すことをＰＲＥ－ＡＣＫされたといい，送信してからＰＲＥ－ＡＣＫされるまでを第１相と呼ぶ。またＰＲＥ－ＡＣＫするために使用した全受信セグメントをEiがＰＲＥ－ＡＣＫすると，条件Ｃ—３が満足され，giはＡＣＫされたという。giがＰＲＥ－ＡＣＫされてからＡＣＫされるまでを第２相と呼ぶ。一方，受信実体Ejが受信セグメントgiを受信し，かつ他の全受信実体がgiを受信したことを認識したとき，gjはEjによりＰＲＥ－ＡＣＫされたという。giが受信されてからＰＲＥ－ＡＣＫされるまでの間を第１相と呼ぶ。また，Ejで受信したセグメントgiをＰＲＥ－ＡＣＫする為に送信したセグメントgi（Gj(i)

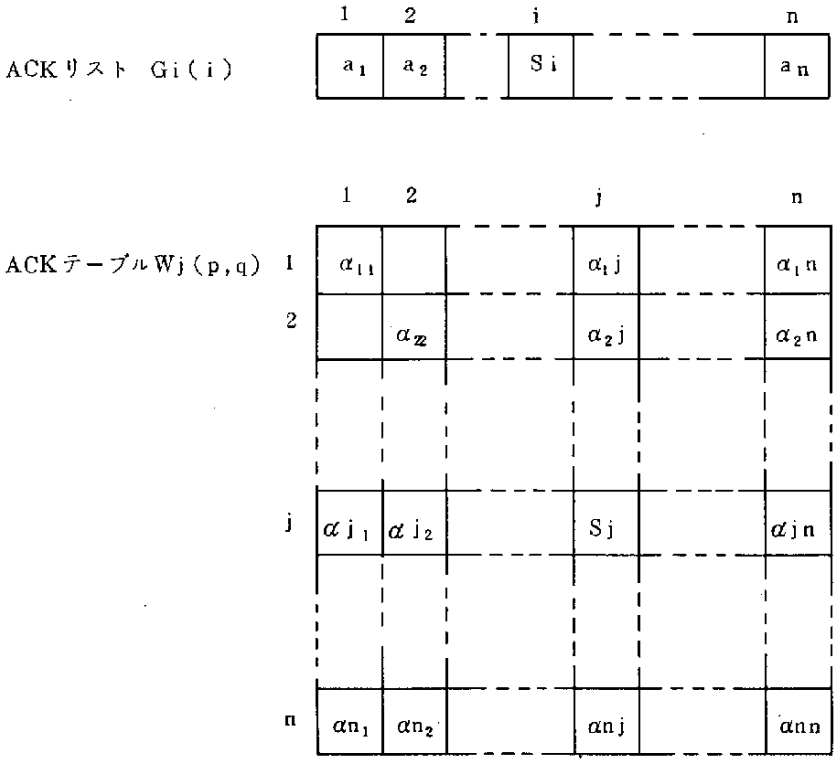


図3.18　セグメントのACKリスト及び実体E_jのACKテーブル

が初めてg_iの通番以上となる）が，PRE−ACKされたとき，g_iはE_jでＡＣＫされたという。g_iがPRE−ACKされてからＡＣＫされるまでの間を第２相と呼ぶ〔図3.19〕。

キューには送信セグメントを格納する送信キュー（ＲＴＱ）及び受信セグメントを格納する受信キュー（ＡＱ）がある。ＲＴＱには第１相のセグメントを格納するＲＴＱ１と第２相のセグメントを格納するＲＴＱ２があり，ＡＣＫされたセグメントはＲＴＱ２から取り除かれる。また，ＡＱにもＲＴＱ同様ＡＱ１及びＡＱ２がある。受信セグメントはＡＣＫされるとＡＱ２から上位層への転送キュー（ＳＱ）へ移され，逐次上位層へ転送される。

実体E_iにおいて，E_jが送信したセグメントg_iが受信セグメントg_jによりPRE−ACKされたとき，g_iがＲＴＱ１からＲＴＱ２へ移る。このg_iとg_jの関係を示すためにg_iからg_jへのポインタaptrを設ける。aptrの指すセグメントg_jがPRE−ACKされ，ＡＱ１からＡＱ２へ移る

と，aptrの元のセグメント（RTQ2にある）がＡＣＫされたことを意味し，ＲＴＱ２から取り除かれる。また，ＡＱ１にある送信セグメントg_jの受信を初めて通知する為にE_iが送信したセグメントg_i（$G_i(j)$が初めてg_jの通番以上となる）とする。このときg_iへのポインタrptrを張る。rptrの元にあるセグメントg_jは他の全受信実体から送られてくるセグメントによりＰＲＥ－ＡＣＫされＡＱ２へ移る。その後，rptrの指すセグメントがＰＲＥ－ＡＣＫされＲＴＱ２へ移ることは，rptrの元のセグメントg_jがＡＣＫされたことを意味し，g_jはＡＱ２から取り除かれＳＱに入られる。

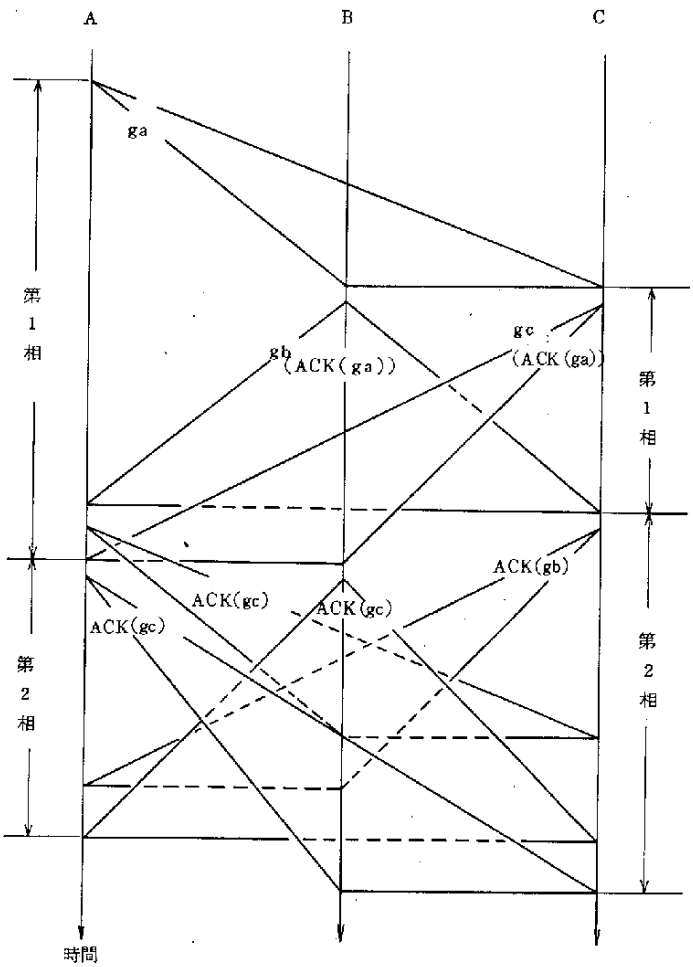
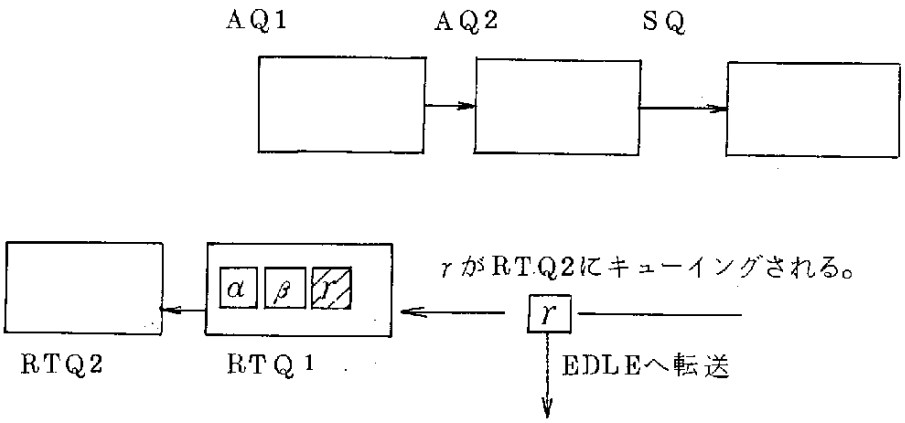


図 3.1 9　2 相による受信確認手順

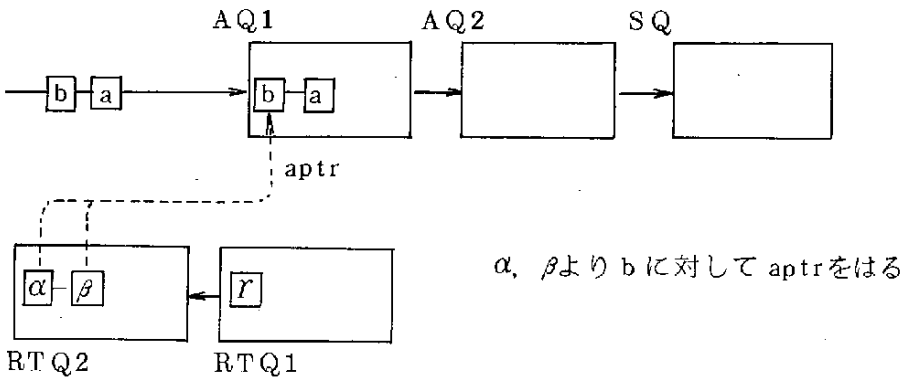
以下，送信セグメント及び受信セグメントのPRE－ACK及びACKの手順を例で示す。

① セグメントα，βがすでに送信されており，受信セグメントはない状態でセグメントγを送信する。

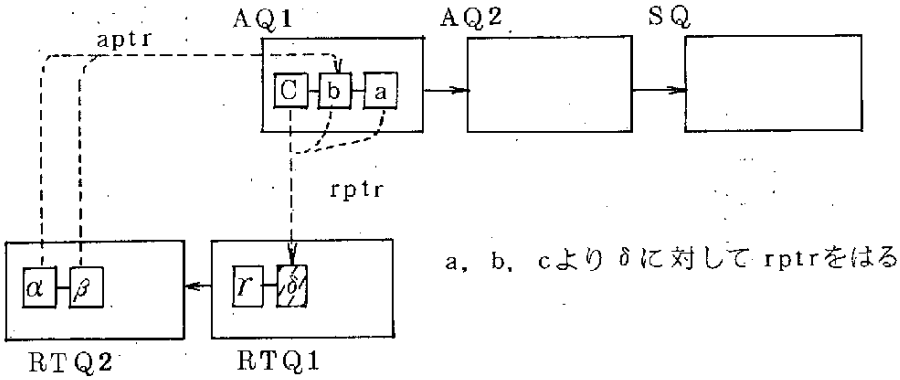


② セグメントa，bが受信され，bの受信によってαとβが全実体で受信されたのが確認された。

（α，βがPRE－ACKされた。PRE－ACKの判定はGとUにより行う）

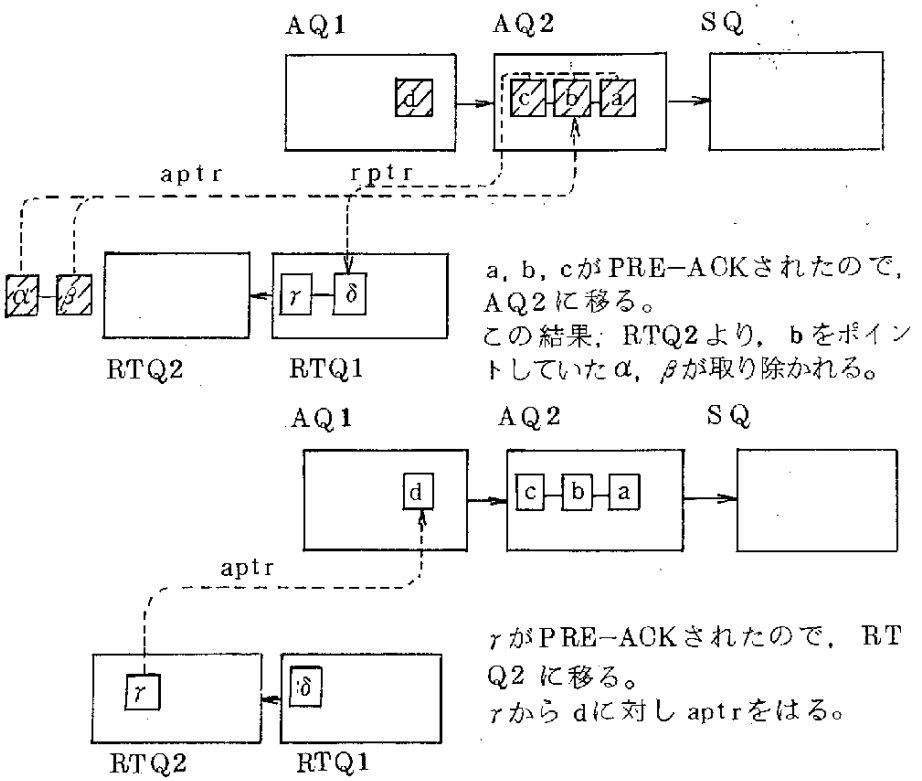


③　Cを受信し，その後δを送信する。δはa，b，cの初めての受信の通知（ack）である。

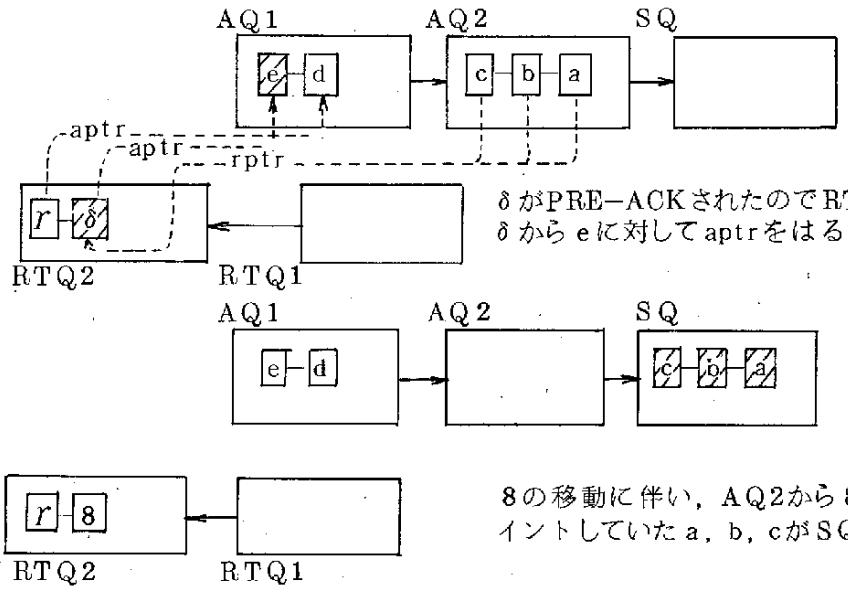


a，b，cよりδに対してrptrをはる

④　dを受信する。dによりa，b，c，γがPRE−ACKされる。



⑤　eを受信する。eにより8がPRE−ACKされる。



キュー操作における第1相は，開設手順におけるSYN－SENT又はSYN－RECに対応する。また，第2相はPRE−ESTABLISHED，ACKされた状態はESTABLISHEDにそれぞれ対応する。開設手順においては，PRE−ESTABLISHEDにおいて全実体からSYN−ACKを受信して始めてESTABLISHEDに移行したが，セグメントの転送においては，通信効率の向上のため，第2相になって一定時間を経過した場合（タイムアウト）ACKされたと等価に取り扱う。これは，第2相に移行した段階で全点が正しく受信されている保障があり，再送要求が発生せず，単純群を保障し得るためである。

(C)　再送要求及び再送

Wをウインド幅としたとき，次の条件において再送要求を行う。

実体E_iで受信したセグメントをg_jとする。g_jのACKリストG_jの要素をa_k，実体E_iのACKテーブル$\mathbb{U}_i$の要素を$\alpha_{pq}(=\mathbb{U}_i(p, q))$としたとき，

E－1　$a_j > \alpha_{ij}+1$かつ，$a_j \leqq \alpha_{ij}+W$。

E－2　$V_k(\neq i)$に対して$a_k > \alpha_{ik}$であるa_kが存在する。

群通信における再送は，単に欠落したセグメントのみの再送では実体間の受信セグメントの順序関係が崩れてしまう恐れがある。この為，リジェクト（ＲＥＪ）手順により欠落セグメント以降の全セグメントを全実体で放棄させ，これらを再送させる。再送要求はＲＥＪセグメントの送信により開始される。ＲＥＪ手順も開設手順と同様ＲＥＪ，REJ－PAK及びREJ－ACK〔表3.1参照〕の3種類のセグメントを用いたＴＢＨ手順により行い，全実体でどこまで受信したかの合意を取る。但し，PRE－REJ CTEDにおいて全実体からREJ－ACKを受信する前にタイムアウトが生じた場合にはREJ CTEDに移行する。また，REJ－SYN，REJ－RECにおいてREJ－ACKを受けた場合も全実体がＲＥＪ又はREJ－PAKを送信したと判断し，PRE－REJECTEDへ移行する〔図3.20〕。

ＲＥＪセグメントg_i内のＡＣＫリストG_iは，実体E_iのＡＣＫテーブル内のα_{iq}（$q=1,\cdots,n$）が使用される。g_iを受けた実体E_jは，E_jのＡＣＫテーブル内のα_{iq}（$q=1,\cdots,n$）とG_iの要素a_q（$q=1,\cdots,n$）をすべてのqについて比較し，$\alpha_{iq}>a_q$であればα_{jq}をa_qで置き換える。この操作で得られたα_{jq}（$q=1,\cdots,n$）を，E_jが送信するREJ－PAKのセグメントg_jのＡＣＫリストG_jとして使用する。各実体は，受信する全REJ－PAKに対して同じ操作を行い，ＡＣＫテーブルを更新する。各実体E_iのＡＣＫテーブル中のα_{iq}（$q=1,\cdots,n$）は，他の全実体からＲＥＪ又はREJ－PAKを受けた段階で群内の最小値に統一される。また，全実体E_iから送信されるREJ－ACKのＡＣＫリストG_iもすべてα_{iq}（$q=1,\cdots,n$）と等しくなる。この結果，単一放送路性及び順序性を保持して再送処理を行うことができる。また，REJ－ACKのＡＣＫリストの各要素a_iで示される通番以降のセグメント（通番a_i+1，$a_i+2\cdots$のセグメント）は各実体E_iのＲＴＱ１内に存在し再送セグメントの存在が保障されている。

ＲＥＪ手順により各キューの操作が行われる。ＲＴＱ２及びＡＱ２内のセグメントはPRE－ACKされているため，各セグメントの通番は必ずREJ－ACKのＡＣＫリストの要素a_i以下となっている。この結果，ＲＴＱ２内の全セグメントは取り除かれ，ＡＱ２内の全セグメントはＳＱに移される。また，REJ－ACKのＡＣＫリストG_iの要素$a_1,\cdots,a_n$

以下の通番のセグメントはＡＱ１及びＲＴＱ１からそれぞれ，ＡＱ２及びＲＴＱ２に移される。この操作と並行して，ＡＣＫテーブルを更新する。ＡＣＫテーブルの要素α_{ij}を，すべてのiに対してa_j($j=1, 2, \cdots,$ ｎ）で置き換える。この結果，群内のすべてのＡＣＫテーブルは等しくなる。

キュー操作及びＡＣＫテーブルの更新が終ると各実体ごと個々にＲＴＱ１からセグメントの再送を開始する。

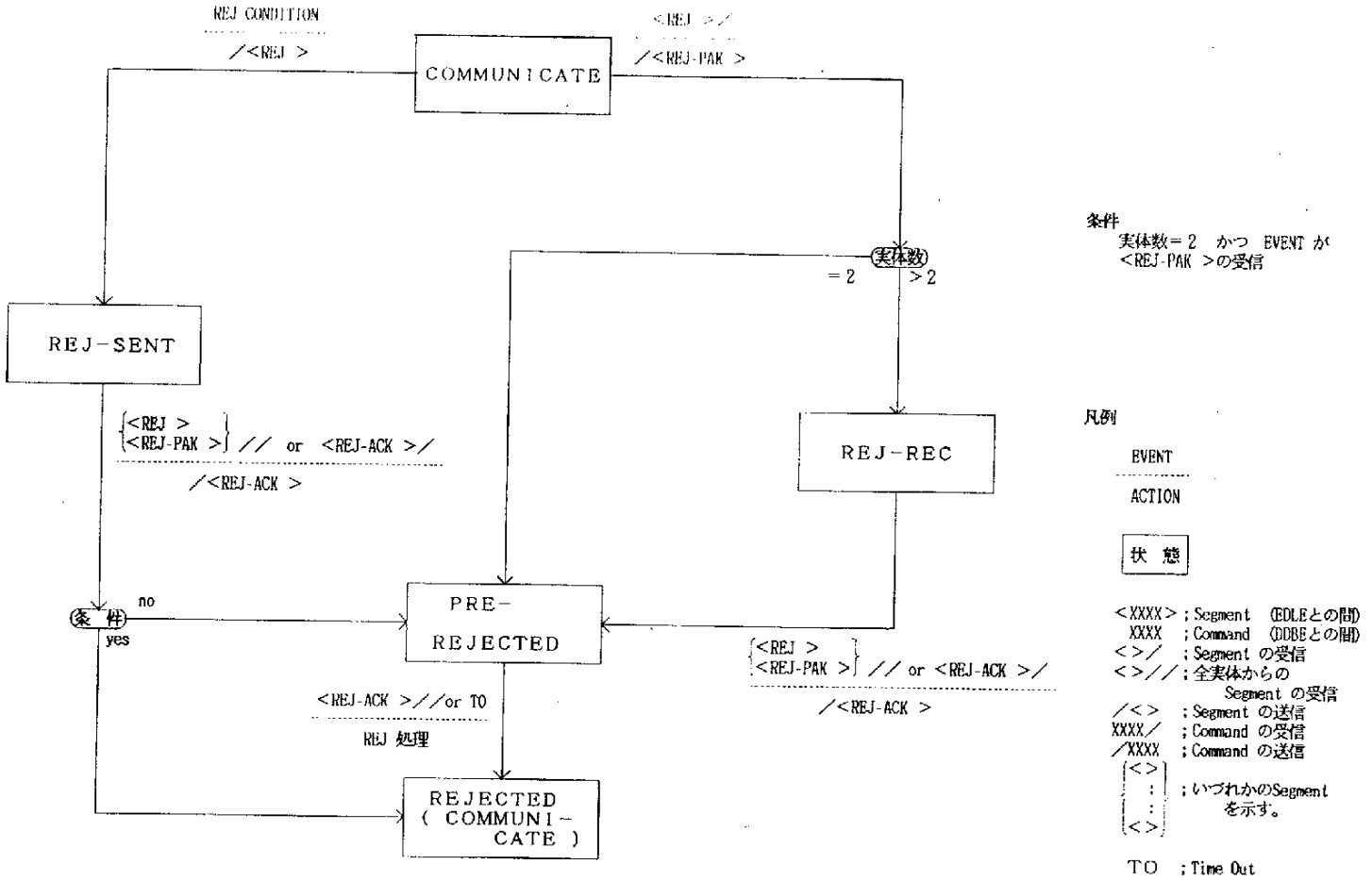


図3.20　ＲＥＪ手順の状態遷移図

C．フロー制御

群通信における実体間のフロー制御は，ウインドウと受信不能通知との２方式を併用する。DARPAのＴＣＰのように受信可能なデータ量をウインドウ幅として他に通知するだけでは群のフロー制御は行えない。これは，各実体が１実体からのデータだけを受信するのではなく，群内の全実体から受信しなければならないためである。

(a)　ウインドウ方式

ウインドウ方式は，システムごとにウインドウ幅Ｗを設定し，各実体はＷ個以上のセグメントの送信を行わないことにより，フロー制行を行う。Ｗ個以上のセグメントとは，ＡＣＫされていないセグメントすなわち，ＲＴＱ１，ＲＴＱ２に存在するセグメントの総数がＷ個を越さないことを意味する。また，このウインドウ幅Ｗは受信通番のチェックにも使用される。

(b)　受信不能通知方式

ウインドウ方式だけでフロー制御を行うと，ｎ実体で構成した群においては，各実体は最大Ｗ×（ｎ－１）個のセグメントを受信しなければならない。ある実体Ｅiで受信バッファが一杯になると，受信不能を示すＲＮＲセグメント〔表3.1参照〕を送信し，ＲＮＲ手順によりＡＱ及びＳＱ内のセグメントの処理を行う。ＲＮＲ手順はＲＥＪ手順とキュー操作も同じであり，異なる点は処理が必要となる原因及び，処理から復帰するタイミングのみである〔図3.21〕。

ＲＮＲ処理からの復帰はＲＲセグメント及びPR−ACKセグメントの送受により同期を取って行う。

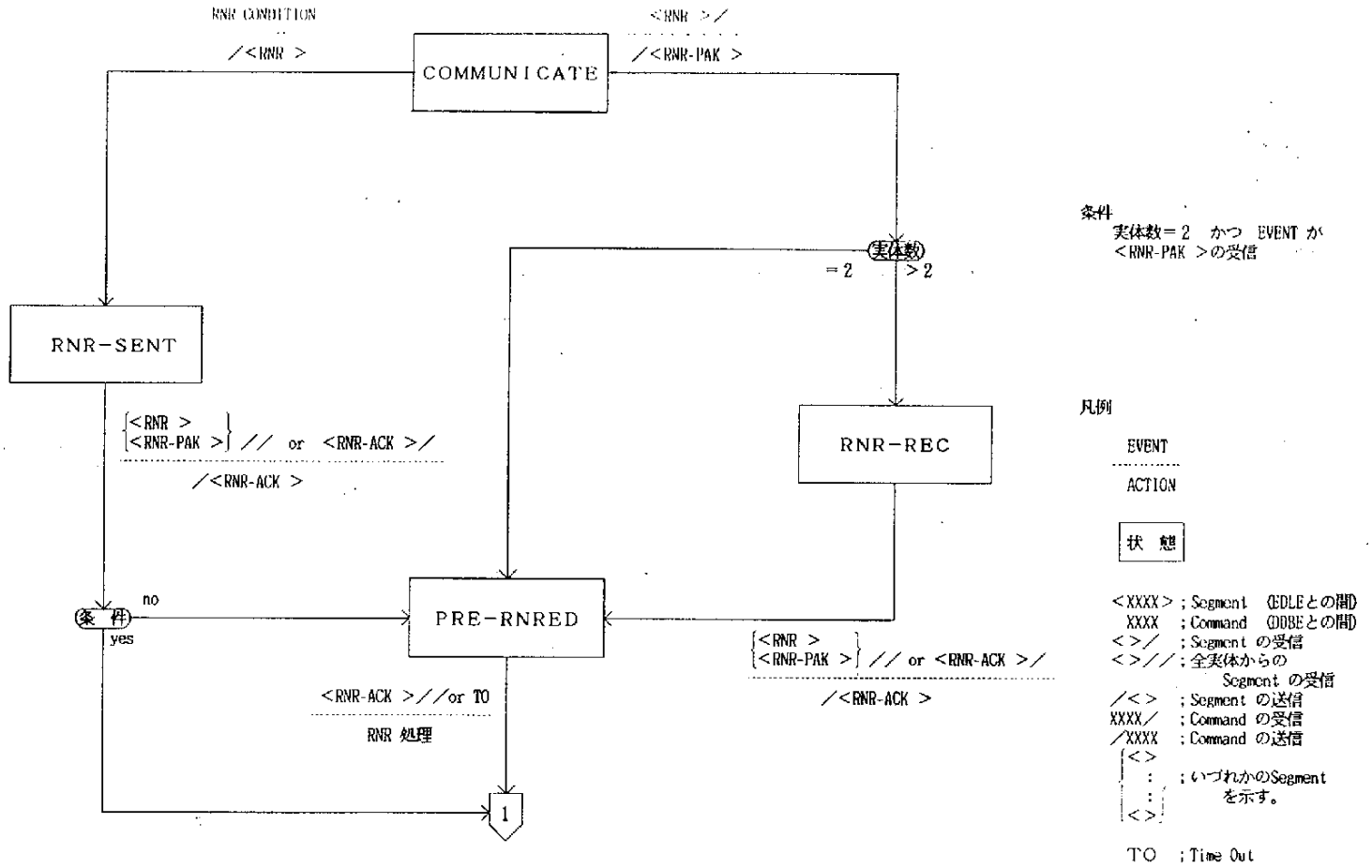


図 3.2.1　ＲＮＲ手順の状態遷移図（その１）

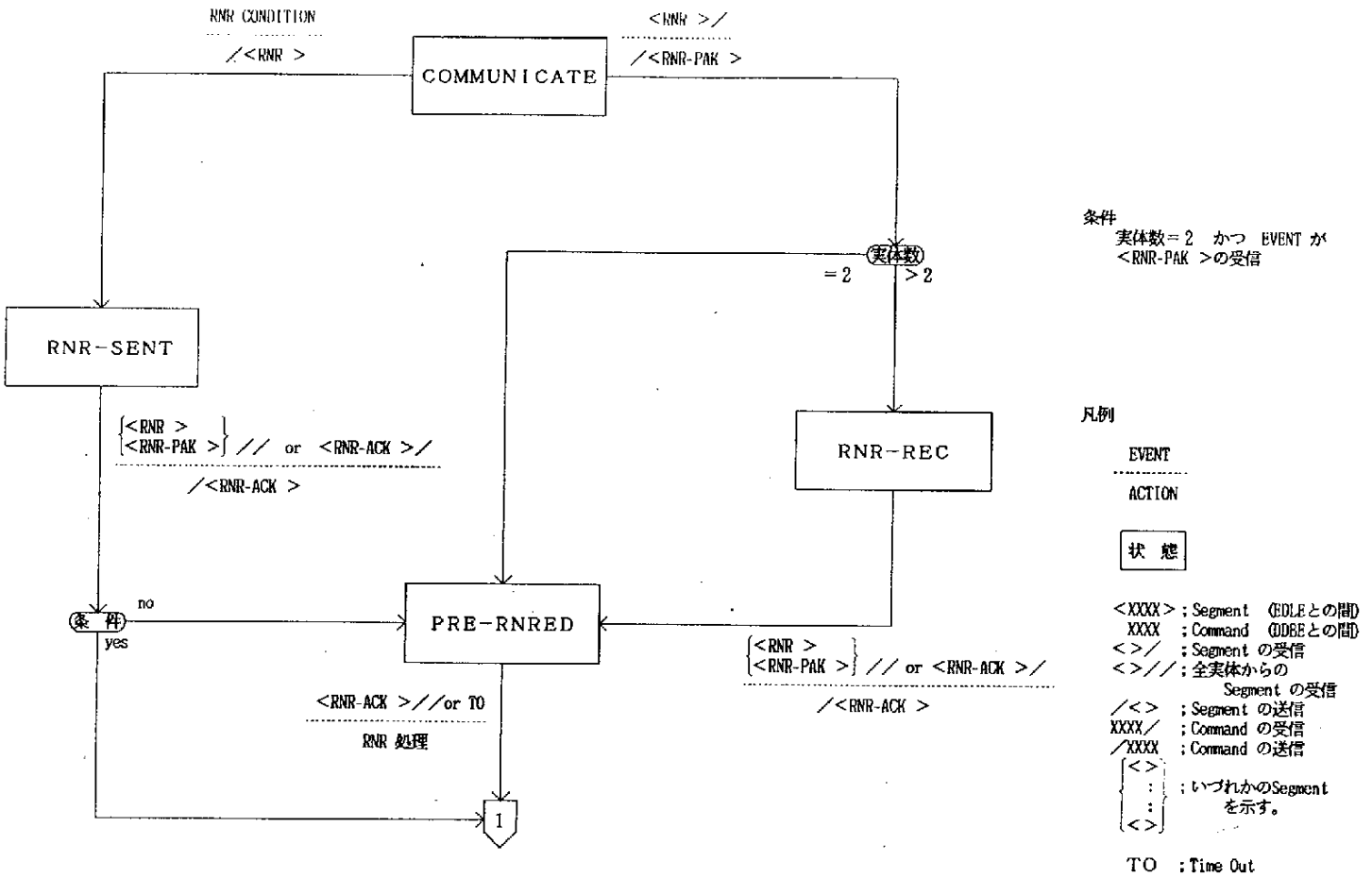


図 3.2 1　ＲＮＲ手順の状態遷移図（その２）

D．群の初期化

データ転送間に再送不可能なエラーが発生した場合，群を構成時の状態に戻すことを群の初期化と呼ぶ。再送不可能なエラーとは，ある実体から受信したセグメントの通番が，その直前に同実体から受信したセグメントの通番よりウインドウ幅W以上隔れた場合である。この場合ＲＳＴ手順〔図3.2.2〕により群を初期化し，上位実体による再送を可能とする。ＲＳＴ手順はＲＥＪ手順と同様に行われる。

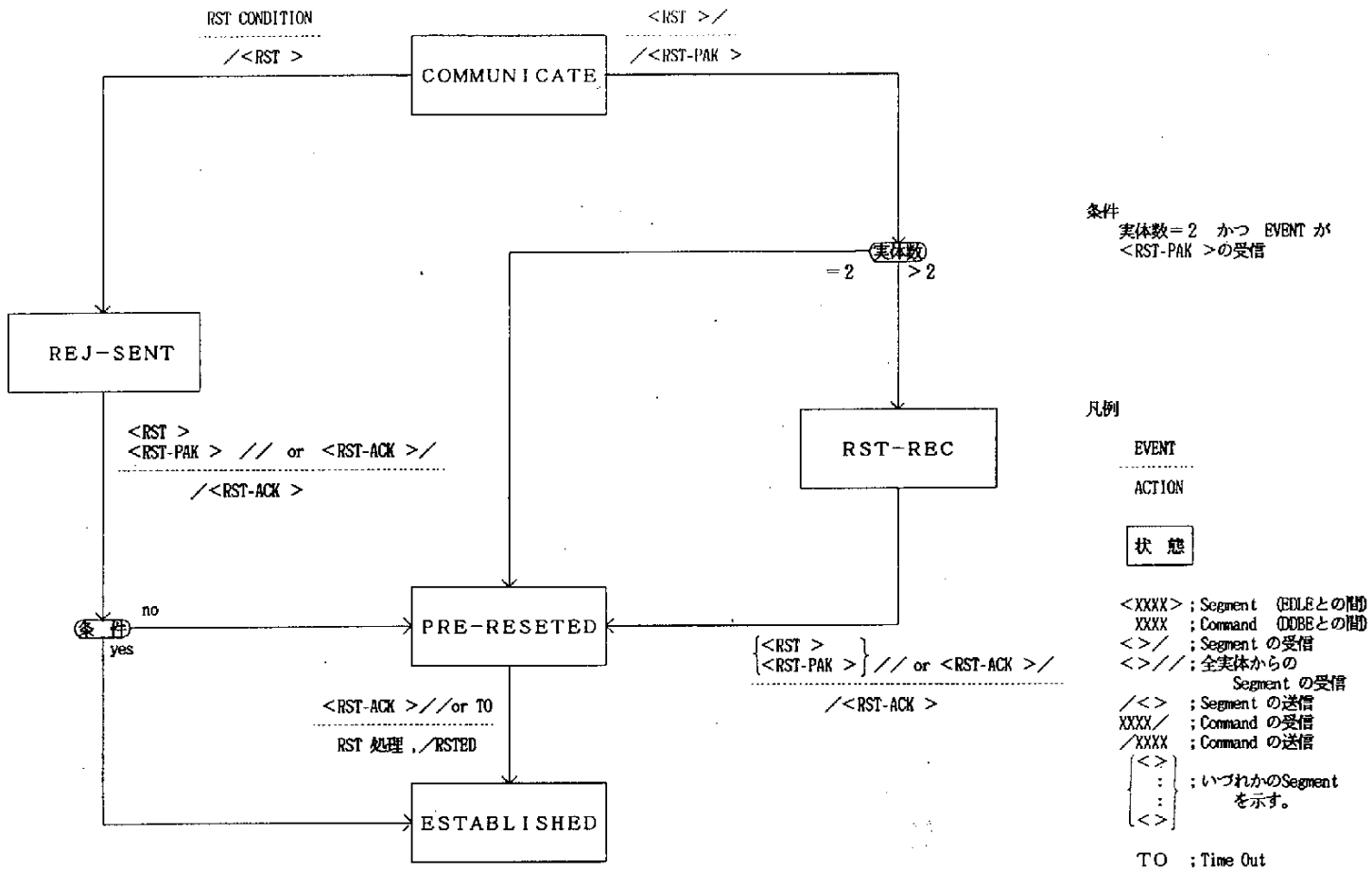


図 3.2.2　ＲＳＴ手順の状態遷移図

E．群の解除

群サービスを提供しているＴＣＣ実体Eiは，ＤＤＢ実体Diからの解除要求を受けると群を解除する。群の解除要求には，正常終了（ＦＩＮ）と異常終了（ABORT）がある。これら解除要求は，DiからＦＩＮ又はＡＢＯＲＴコマンドのフレームにより通知される。

(A)　ABORT

ABORTは上位層において何らかの異状が発生した場合に受ける解除要求で，群通信の状態のいかんにかかわらず群を強制終了させる。ＡＢＯＲＴコマンドを受けたDiはABORTセグメントを送信する。ABORTセグメントを受けたＴＣＣ実体Ejもまた，ABORTセグメントを送信する。Ei，EjともにABORTセグメント送信後，下位実体にＣＬＳＤコマンド，上位実体にABORTコマンドを送り，群サービスを終了する。

(B)　ＦＩＮ

ＦＩＮは，ＤＤＢ実体Eiが処理を終了し，Diからの送信データがなくなった段階で発する解除要求で，群内の全通信の終了を待って群が解除される手順である。

DiからＦＩＮコマンドを受けるとEiは，Diから受けた送信データの最後のデータを送信するためのセグメントをＦＩＮセグメントとして送信する。また，各実体Ejは通信中ＦＩＮセグメントを監視する。各実体は自分を含む全実体からＦＩＮセグメントを受信し，かつ全ＦＩＮセグメントがＡＣＫされた段階で，上位及び下位実体にＣＬＳＤコマンドを送り群サービスを終了する。

3.2.5　群への加入及び退会

群への加入とは，開設されている群にＴＣＣ実体が新たに加わることである。加入には，現在構成されている群を認識する必要がある。一方，通信期にある群は，ＣＩＤのみで識別されているため，加入を実現するためには各ＴＣＣ実体が常に網内の群の状態を管理する必要がある。また，群の構成員が増加するとセグメント内及び各ＴＣＣ実体内のＡＣＫテーブルの変更が必要となり，群への加入手続は現在行われている通信を中断して行う必要性が生じる。

群からの退会とは，群の構成員のうちある実体のみが抜けることで，加入

と同様にＡＣＫテーブルの変更等を必要とする。

また，群への加入及び退会は，加入又は退会しようとする実体だけの要求により行い得るものか，あるいは群の構成員に許可を受ける必要があるか，あるとするとだれから受けるか等，許可の問題が生じる。この許可は，上位プロトコルであるＤＤＢプロトコル及びＤＤＢサービスの内容によるところが大きい。

群への加入及び退会は今後，その必要性，可能性及び群サービスの効率等を考慮して検討すべき問題である。

3.3　トランスポート群制御プロトコルの実現

この節においては，ＬＩＳを構築する機器の構成（ローカルエリアネットワーク）に基づき，トランスポート群制御プロトコル（ＴＣＣＰ）の実現について記述する。

3.3.1　開発環境及びＬＩＳの基本構成

ＬＩＳは，同軸型ＬＡＮ（Ethernet型）及びＬＡＮに接続された大型コンピュータ，ミニコンピュータ並びにスーパーパソコン等で構成される分散システム内に構築されるソフトウェア（プロセスの集合）から成る。ＬＩＳの開発において使用する機器は，Ethernetと等価なＥＤＬサービス(Ethernet Data Link Service)を提供するNet／One（アンガマンバス社）及び大型コンピュータＭ－360 R（富士通），ミニコンピュータU－1200（パナファコム）並びにスーパーパソコンSUN－2／120（ＳＵＮマイクロ社）から成る〔図3.2.3〕。また，ＬＩＳの開発は，ＥＤＬＳ及び各ホスト上のＵＮＩＸオペレーティングシステムを用いて行い，各実体はＵＮＩＸ上のプロセスとしてＣ言語で実現されている。実体間の通信はＵＮＩＸの提供するプロセス間通信機能を利用する。ＵＮＩＸは現在，ＡＴＴ版とバークレー版があり，U－1200はＡＴＴ版のＳＹＳＴＥＭⅢ，ＳＵＮはバークレー版4.2ＢＳＤをそれぞれＯＳとして使用している。4.2ＢＳＤではプロセス間通信にＴＣＰ／ＩＰプロトコル〔DARPA81〕が使用できるが，ＳＹＳＴＥＭⅢではこの機能がなく，パイプと呼ばれる機能（システムコールpipe）のみがある。このため，プロセス間通信は，現在パイプを利用して行っている。パイプは，ある親プロセスが，フォーク（システムコールfork）により生成した子プロセ

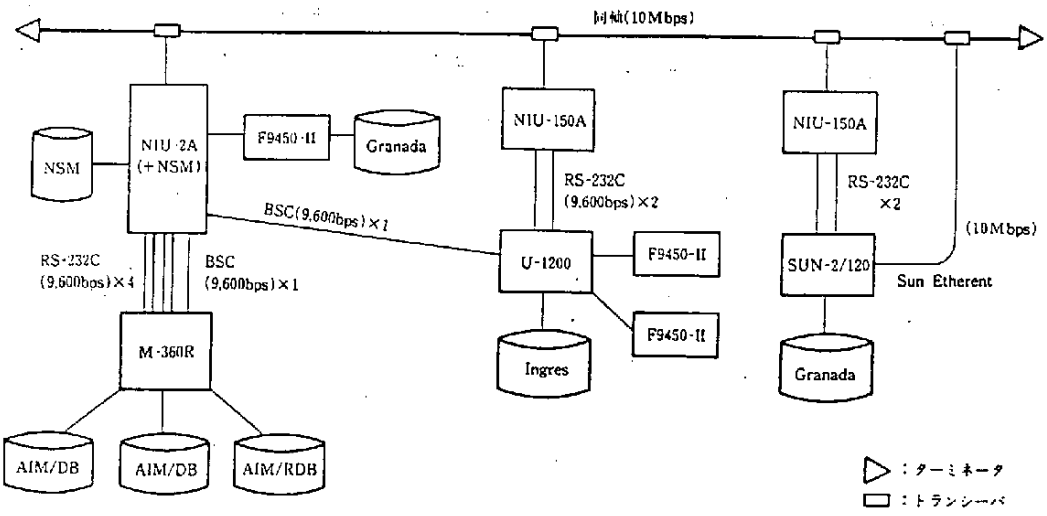


図3.2.3　ＬＩＳの構成

スとの間にのみ構成できる。また，パイプによる通信は，ファイルのread, writeと同等で，フイル管理テーブル上にファイルディスクリプタ（ＦＤ）が，１つのパイプに対して設定される。１個のプロセスは最大２０個までのファイルディスクリプタを使用できる。この結果，ＴＣＣＰにおいては多重度最大3の群通信サービスの提供を目標に設計を行った。

ＥＤＬサービスはNet／Oneにより提供される。ＬＩＳを実現する際，ＥＤＬ実体は，Net／Oneのコントローラ，ネットワークインターフェースユニット（ＮＩＵ）内にあり，ＥＤＬ層以下をブラックボックスとして取り扱うことができる。このため，ＥＤＬ実体とＴＣＣＥ実体のインターフェース（EDL－SAP）をＮＩＵのドライバー，回線（RS－232C，GP－IB等）及びホストのドライバ，ＥＤＬ実体をＮＩＵとそれぞれ見なすことができる。ＮＩＵを制御するコマンドをＥＤＬフレームとして取り扱うことができる〔表3.3〕。以上，本節においてＮＩＵはＥＤＬ実体と等価であるものとして記述する〔図3.2.4〕。

各プロセスの関係及び起動の要領を図3.2.4, 3.2.5に示す。この図はプロセス間通信にパイプを使用することが前提となっており，ＴＣＰ／ＩＰを使用する場合はより簡略化される。ＵＮＩＸシステムにおいては，コマンドイン

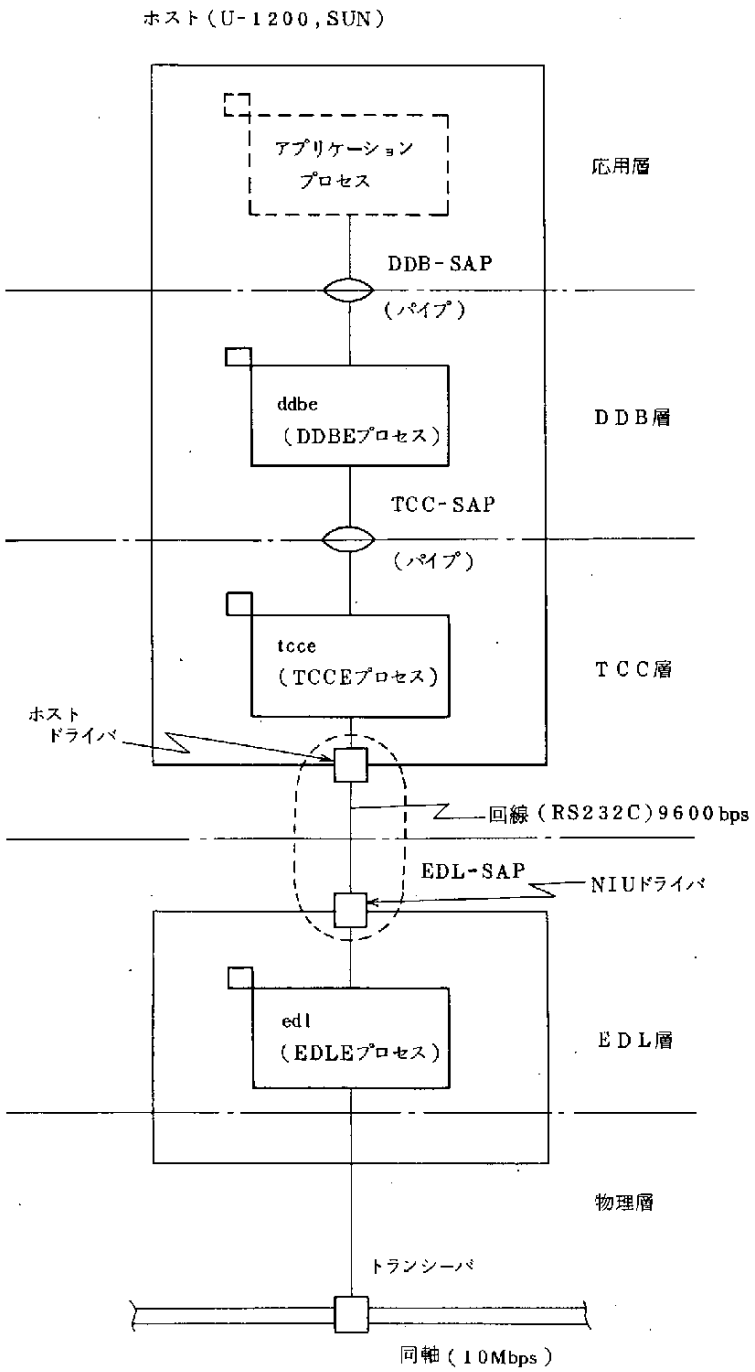


図 3.2 4　各プロセスの関係

ターブリターcshが起動され，コマンド待ちの状態になっている。このときコマンドとしてプロセス名main（以下，プロセス名を英小文字で表わす）を入力するとmainが起動する。mainでは必要数のパイプを構成し，フォークして子プロセスを生成起動する。このとき子プロセスとの間にパイプが構成される。次に親mainは，コマンドインタープリターshを起動する。もし，cshを起動すると，各パイプのファイルディスクリプターが消去される。一方，子mainはＴＣＣＰ実行モジュールのもととなるlisプロセスを起動する。lisプロセスはフォークして子lisプロセスを生成する。この段階でsh，親lis及び子lisのそれぞれの２プロセス間にパイプが構成されている。構成されたパイプのうち不要（図中のpipe 11, 21, 31は同一のメモリーを共用するものでプロセス間通信を正しく行うためには，３個のパイプのうち２個を閉じる（close）必要がある）なパイプを閉じる。その後，親lisはＴＣＣ実体に対応するtcceプロセス，子lisはＤＤＢ実体に対応するddbeのプロセスを起動する。この段階においてＬＩＳはshからのアプリケーションプロセスの起動要求のコマンド入力待ち状態になる。起動されたアプリケーションプロセスにはddbeプロセスとの間にパイプが構成されており，このパイプからddbeに対し各種処理要求を行う。一般に，起動されるプロセスはデータベースの利用プロセス（クライアントプロセス）であり，サーバープロセスはddbeにより起動される。

3.3.2　トランスポート群制御機能の実現

ＬＩＳ階層モデルにおいては，ＥＤＬ実体がＴＣＣ層に複数のサービスアクセス点（ＳＡＰ）を提供することにより，群通信の多重化が実現される。しかし，トランスポート群制御プロトコル（ＴＣＣＰ）の実現においては，Ethernet データリンクサービス（ＥＤＬＳ）をそのまま利用する。また，ＥＤＬＳはＮＩＵ内にあり，ＥＤＬＳが複数のEDL－SAPを提供する構造を構築することは困難である。このため，ＴＣＣＰの実現において，ＮＩＵとホスト間のリンク制御及び群の多重化に必要な各種の機能を行うＤＬ実体（ＤＬＥ）を，ＴＣＣ層に構成する。また，トランスポート群制御（TCC）機能を実現するため，２種類の実体ＴＣＣＥ－Ｓ及びTCCE－Pを用いる。これは，ＣＰＵの使用効率を上げるためで，図3.26に示めす構成で常時多重度３の群サービスを提供するためのプロセスを動かすことは，ＣＰＵの使

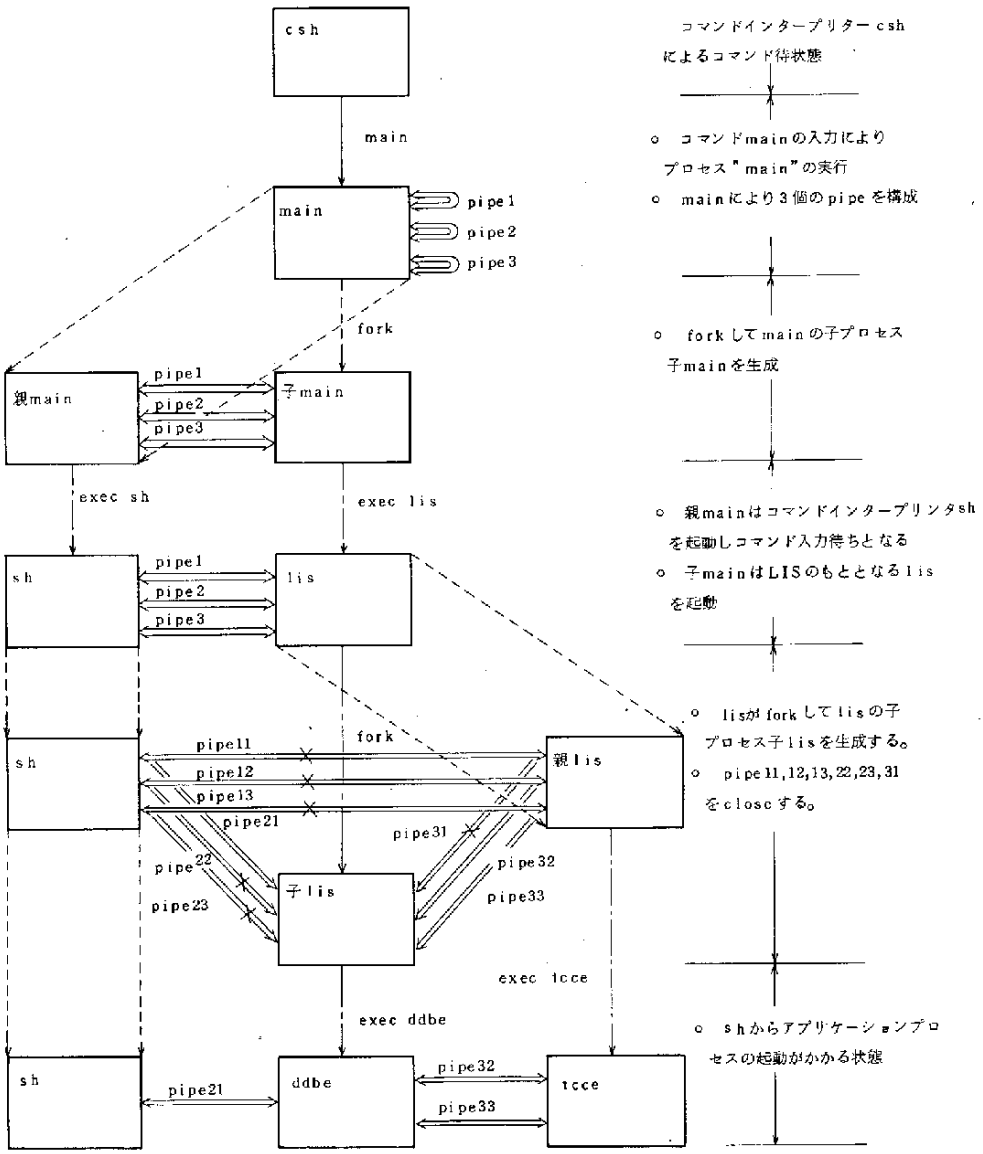


図3.25　ＬＩＳのプロセス起動

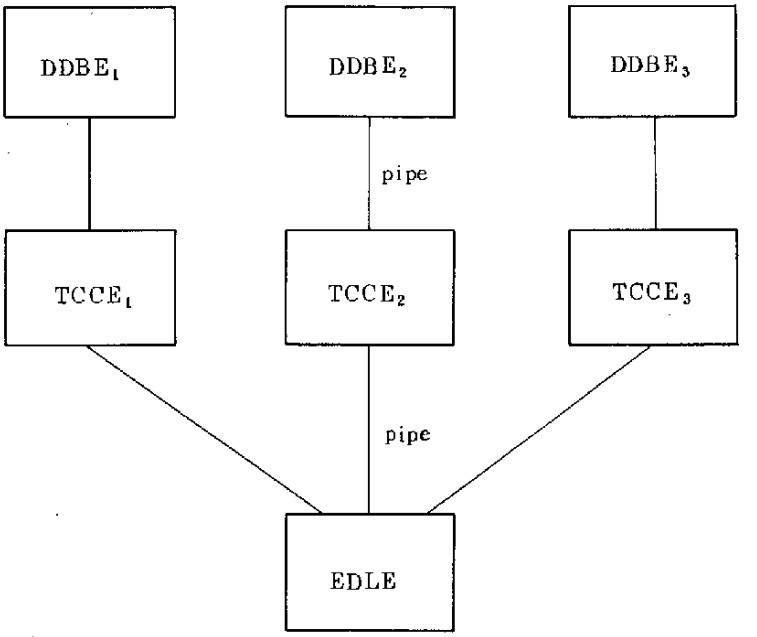


図3.26 多重度3の実体構成

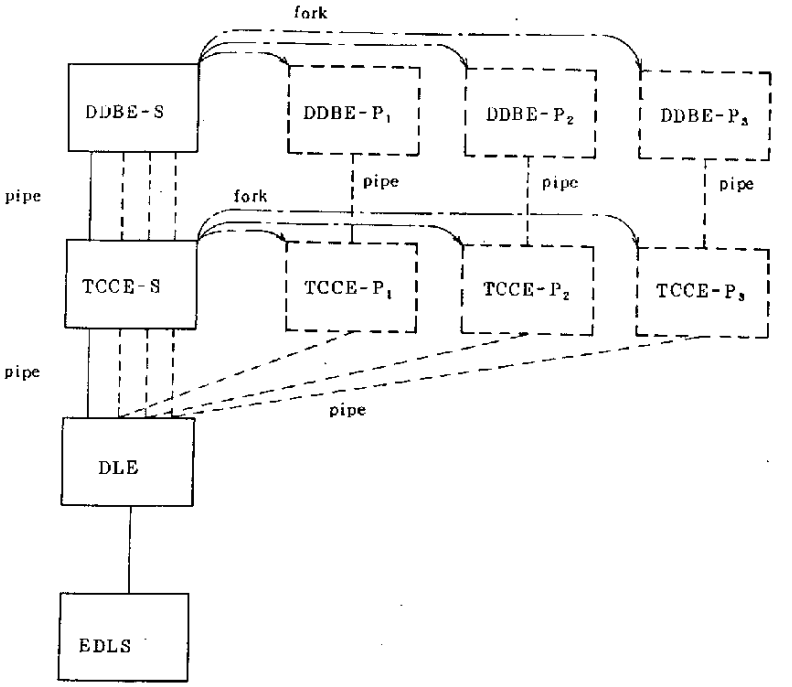


図3.27 DLE, TCCE—S, TCCE—P

用効率の低下，群サービスの効率の低下につながる。今回行ったＴＣＣＰの設計においては，図3.２７に示す構成を採用した。この方式においては，ＤＤＢ実体から開設要求を受け，必要時にTCCE−Pを起動する。群通信に必要な各種機能は，ＴＣＣＥ−Ｐにより実現される。この結果，ＤＤＢＥについても，実質的にＤＤＢＳの処理を行う実体DDBE−Ｐ及び利用者とのインターフェースとなりDDBE−Ｐの起動，群の開設要求等を行う実体ＤＤＢＥ−Ｓの２種類のＤＤＢ実体が必要となる。

各プロセスの起動要領は，図3.２５の手順と同様であり，各プロセス，sh間にそれぞれ４本のパイプを構成する。また，図3.２５のプロセスtcceから図3.２８の要領でTCCE−Ｓ，ＤＬＥ及びTCCE−Ｐを起動する。図3.２５

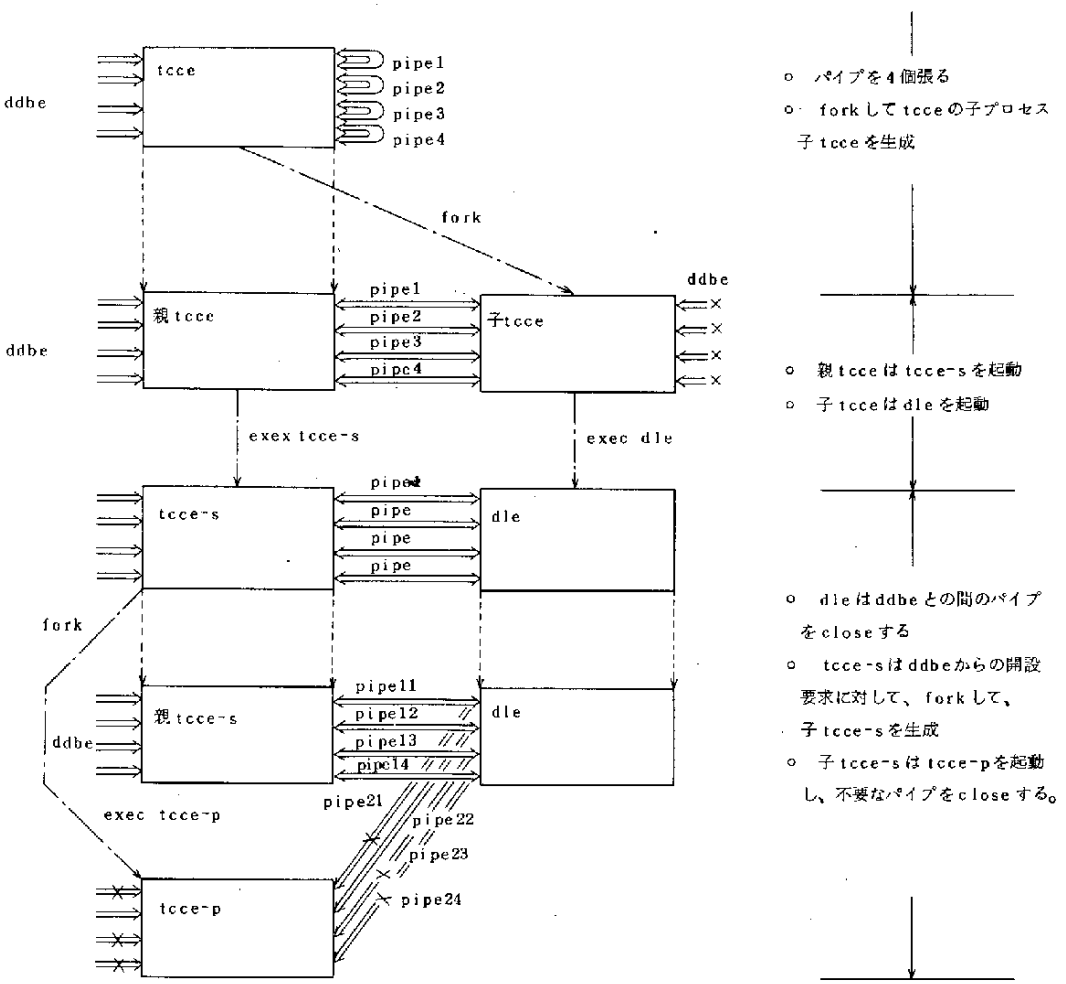


図3.２８　TCCES，DLE及びTCCE−Pの起動

の最終段階においてプロセスtcceはパイプを4個構成した後子プロセス，子tcceを生成する。子tcceはddbeとはパイプをも有するため，これを閉じた後，ＤＬＥの機能を実現するプロセスｄｌｅを起動する。一方，親tcceはＴＣＣＥ－Ｓの機能を実現するプロセスtcce－ｓを起動する。この状態でtcce-sはｄdbeからの開設要求を待つ。tcce-sは開設要求を受けるとforkして子プロセス子tcce-sを生成する。子tcce-sは，プロセスddbe及びｄｌｅに対してそれぞれ4本のパイプを有している。実際に使用するパイプはそれぞれ1本であり，子tcce-sが生成された段階で，tcce-sがfork時に指定したパイプを残し，それ以外のパイプを閉じる。パイプの処理が終ると，子tcce-sはＴＣＣＥ－Ｐの機能を実現するプロセスtcce-pを起動する。

3.3.3　各実体間のインターフエース

各実体間のインターフェースとして，Net／One（ＮＩＵ）とＤＬＥ（Ｎ－Ｄ），ＤＬＥとＴＣＣＥ－Ｓ（Ｄ－Ｔｓ），ＤＬＥとＴＣＣＥ－Ｐ（Ｄ－Ｔｐ），ＴＣＣＥ－ＳとＤＤＢＥ－Ｓ（Ｔｓ－Ｂｓ），ＴＣＣＥ－ＰとＤＤＢＥ－Ｐ（Ｔｐ－Ｂｐ）及びＴＣＣＥ－ＳとＴＣＣＥ－Ｐ（Ｔｓ－Ｔｐ）の6種類がある。Ｎ－Ｄインターフェースは RS－232Ｃ又はＧＰＩＢを用い，ＤＬＥフレームにより行われる。また，Ｔｓ－Ｔｐ インターフェースはＴＣＣＥ－Ｐのプロセスが起動するときにのみ，ＢＴＣファイル（後述）を用いて行われる。ＢＴＣファイルは，ＴＣＣＥ－Ｐで作成され，ＴＣＣＥ－Ｐが読んだ後削除する。その他のインターフェースはパイプ（ＵＮＩＸバージョン 4.2 ＢＳＤではＴＣＰ／ＩＰが使用できる）を用い，ＴＣＣフレーム〔表3.4〕により行う。Ｄ－Ｔｓ及びＴｓ－Ｂｓ では管理用コマンドの各ＴＣＣフレームまたＤ－Ｔｐ及びＴｓ－Ｂｓ では通信用のコマンド及びデータの各ＴＣＣフレームをそれぞれ用いる。〔表3.4〕。

表3.4　DDBE，TCCE－S，P及びDLE間のフレーム一覧表

コマンド	コード	DDBE---TCCE	TCCE---DLE	種別	フレームデータの有無	機能
C-READY	1		←	管理	有	LIS 通信処理の起動時EDLEがTCCEに送る同期フレーム
C-ACK	2	←	←	管理	有	新たな群の開設要求に対する許可応答。
C-NAK	3	←	←	管理	無	新たな群の開設要求に対する拒否応答。
C-AOPEN	4	→	→	管理	有	新たな群の能動的な開設を要求する。
C-POPEN	5	→	→	管理	有	新たな群の受動的な開設を要求する。
C-CLOSE	6	→	→	管理	無	LIS 通信処理の終了要求。（正常終了）
C-CLSD	7	←	←	管理	無	LIS 通信処理が終了したことを示す、C-CLOSE に対する応答。（正常終了）
C-RESET	8	→	→	管理	有・無	各階層の初期化、または開設中の群を終了させる。
C-ABORT	9	←→	←→	管理	有	LIS 通信処理の異常終了要求及び応答。（上位から下位は要求，下位から上位は応答）
C-STATE	10	←→	←→	管理	有・無	LIS 通信処理の状態の確認要求及び応答。（上位から下位は要求，下位から上位は応答）
DATA	20	←→	←→	通信	有	プロトコルデータ単位の上下実体間での転送に使用。
READY	21		←→	通信	有	新たな群を開設する際、群を構成するTCCEとEDLEの間で最初に授受する同期用フレーム。
OPND-CID	22		→	通信	有	群識別子をEDLEに通知する。
CLSD	23	←	→	通信	無	群通信の正常終了の通知。
ABORT	24	←→	←	通信	有	群通信の異常終了の通知。
RETRY	25		→	通信	有	群開設時に開設の再行をEDLEに通知する。
STATE	26	←→	←→	通信	有・無	群の状態の確認要求及び応答。（上位から下位は要求，下位から上位は応答）
ACK	27	←		通信	無	DDBEから受けたストリームの送信が完了し群内の全DDBEで受信されたことの通知。
OPND	28	←		通信	無	群の開設が完了したことをDDBEに通知。
FIN	29	←→		通信	無	群の解除を開始することの通知。
RNR	30			通信	無	未使用
RR	31			通信	無	未使用
RSTD	32	←		通信	無	群通信の初期化（群が開設された時点の状態）が行われたことの通知。

3.3.4 ＤＬＥ

ＤＬＥの役割として次の４つがある。

(i) 多重化の実現

(ii) Net／Oneとのインターフェース

以下，各役割の概要を記述する。

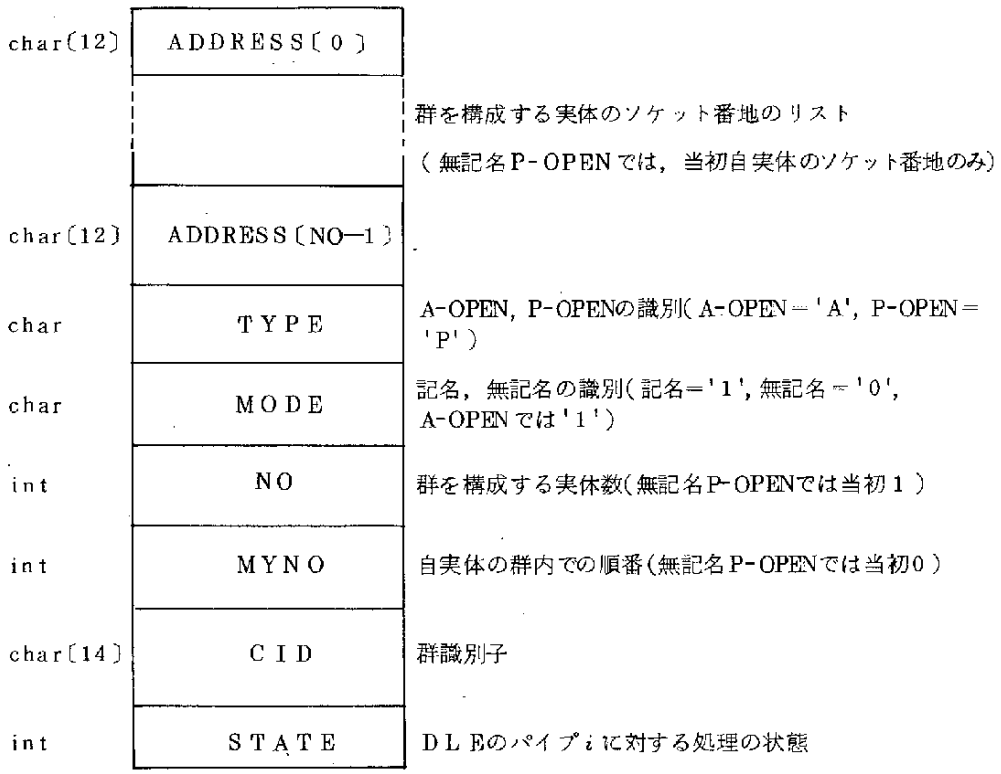
Ａ．多重化の実現

多重化を実現するため，ＤＬＥは次の処理を行う。

〔Ｐ－１〕 パイプを管理し，群の開設要求に対しパイプを配当する。

〔Ｐ－２〕 Net／One（ＮＩＵ）から受信したセグメントを群識別子（ＣＩＤ又はＯＣＩＤ）の指定に基づき，対応するＴＣＣＥ－Ｐへ送る。

Ｐ－１の処理は，ＴＣＣＥ－Ｓとの間でパイプ０を介してコマンドのＴＣＣフレーム（以下，本節ではＴＣＣフレームを単にフレームと呼ぶ）の送受に基づき行われる。ＴＣＣＥ－Ｓとの間で使用されるフレームは管理用〔表3.4〕のコマンドフレームである。ＤＬＥはＴＣＣＥ－Ｓから開設要求を受けると，未使用パイプを選定し，そのパイプ番号i（i＝1，2，3）をＴＣＣＥ－ＳにフレームＣ－ＡＣＫに格納して通知する。このパイプiが，実質的なＥＤＬ－ＳＡＰとなり，群を開設するために，新たに起動されるＴＣＣＥ－Ｐとの間の通信に使用される。一方，未使用パイプがない場合には，フレームＣ－ＮＡＫにより開設要求を拒否する。ＤＬＥはパイプの配当時，パイプ管理テーブル〔図3.29〕を作成する。このパイプ管理テーブルにより，群通信とパイプの対応を図る。ＤＬＥは，Net／Oneが提供するEthernet の放送通信サービスを受ける。この放送通信サービスでは，同軸上に送信されたすべてのパケット（Ethernet パケット）を受信し，ＤＬＥに送る。このため，ＤＬＥには，上位のいずれのＴＣＣＥ－Ｐも構成員となっていない群通信のセグメント及び上位のＴＣＣＥ－Ｐが送信したセグメントまでもがＮＩＵから送られてくる。ＤＬＥは，これらのセグメントの群識別子（ＣＩＤ）又は開設期群識別子（ＯＣＩＤ）内のソケット番地のリスト（S_1，…，Sn）及び実体番号を比較し，不安なセグメントの破棄及び必要なセグメントの対応するＴＣＣＥ－Ｐへの転送を行う〔付記図３パイプiの状態遷移図〕。



int；整数型（システムによりバイト長は異なるSUN：4Byte，U-1200：2Byte
char；文字型（1Byte）
char〔n〕；n個の文字型の配列

図3.29　パイプ管理テーブル

B．Net／Oneとのインターフェース

ＤＬＥは，Net／One（ＮＩＵ）とのインターフェースの役割を果し，ＥＤＬサービス（ＥＤＬＳ）を受けるため，次の処理を行う。

〔Ｐ－３〕　ＮＩＵの状態を制御するための各種コマンド（ＥＤＬフレーム）の送受〔表3.3〕。

〔Ｐ－４〕　TCCE－PとEDLE（NIU）との間でセグメントの送受を行うためデータ用のＥＤＬフレーム"Here Is Data"（以下ＤＬＥフレームを"×××"で記す）の作成及びデータ用のＥＤＬフレームからのセグメントの抽出を行う。

Ｐ－３の処理は，ＤＬＥプロセス（dle）が起動してから終了するまでの間に逐次ＮＩＵとの間でコマンドのＤＬＥフレームを用いて行われる〔付記図２ＤＬＥ状態遷移図〕。

(a)　ＮＩＵの初期化

ＤＬＥからＮＩＵに対して"Clear－NIU"を送り，ＮＩＵからの応答"Cleared"を受ける。

(b)　ホスト番地の取得

ＤＬＥはホスト番地を知るために，ＮＩＵに対し"Who Am I"を送り，応答"You Are"を受ける。また，ＤＬＥは"You Are"内のホスト番地をフレームのＣ－READYを用いて，TCCE－Sに通知する。このホスト番地の通知は，ＤＬＥとＮＩＵの間の通信が確立し，ＥＤＬＳを受けられる状態になったことのTCCE－Sへの通知でもある。

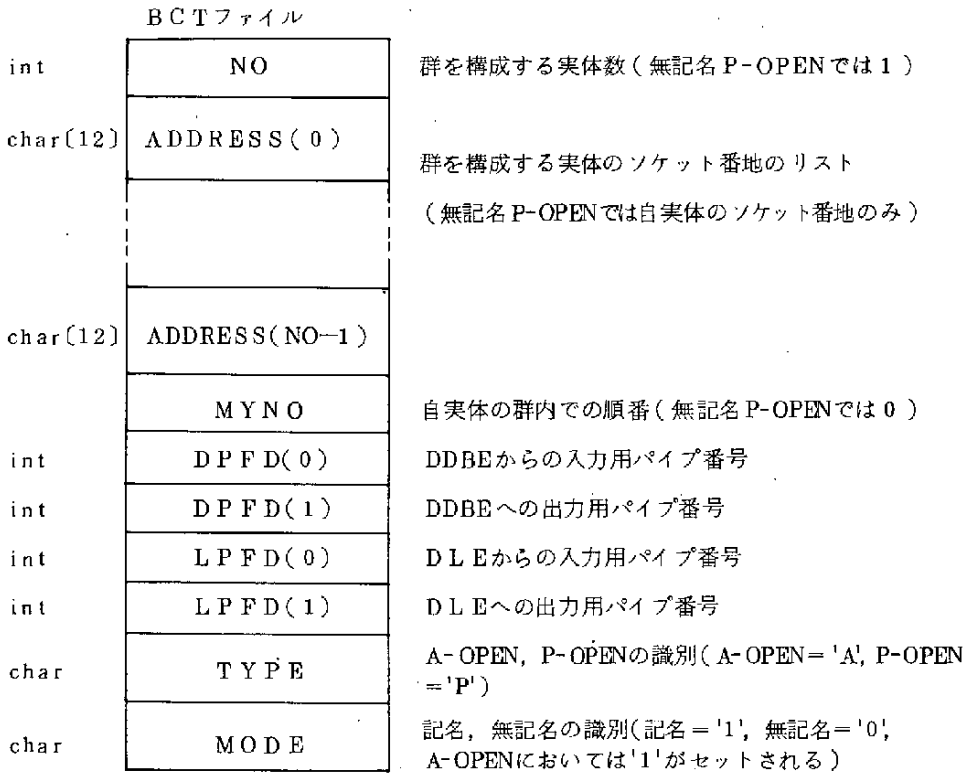
(c)　ＮＩＵの状態の確認

ＮＩＵから一定時間受信ＤＬＥフレームがない場合，"NOP－Request"を送り，ＮＩＵからの応答"NOP－Reseponse"の受信を待つ。ある時間（システムごとに設定するタイマー値）内に応答があればＮＩＵが正常状態にあると判定する。また応答がない場合は，ＮＩＵの初期化から処理を再行する。

(d)　その他の制御

ＮＩＵから"ＮIU－Reset"をＤＬＥが受けた場合，DLEは"NIU－Reset－Seen"をＮＩＵに送る。ＤＬＥはＮＩＵに対しコマンドのフレームを送る際タイマを起動する。タイマ値で与えられた時間が経過しタイムアウトになっても応答がない場合，ＤＬＥはＮＩＵの状態確

認（(c)項）の処理を行う。



int；整数型（システムによりバイト長は異なる　SUN：4Byte　U-1200：2Byte）
char；文字型（1Byte）
char〔n〕；n個の文字型の配列

図3.30　BCTファイルのフォーマット

3.3.5　TCCE−S

TCCE−Sの役割は，DDBE−Sから受けた開設要求に基づき，TCCE−Pのプロセスを起動させることである。このため，DLEに開設要求C−AOPEN又はC−POPENのフレームを送り，許可応答C−ACKを待つ。TCCE−SはC−ACKを受けるとBCTファイル〔図3.30〕を作成し，TCCE−Pのプロセスを生成・起動させる。また，DDBE−Pに対し許可応答を送る。この許可応答には，新たに生成されるDDBE−P及びTCCE−Pの間で使用するパイプ（実質的にこのパイプが群端点CEPとなる）の番号

が格納されている。一方，開設要求に対して，ＤＬＥが拒否応答Ｃ－ＮＡＣを送ってきた場合，ＴＣＣＥ－ＳもＤＤＢＥ－Ｓに対して拒否応答を行う。

また，ＴＣＣＥ－Ｓは，ＤＤＢＥ－Ｓから送られてくるＬＩＳの管理用コマンドに対し必要な処理をＤＬＥと協同して行う〔付記図4. ＴＣＣＥ－Ｓ状態遷移図〕。

3.3.6　TCCE－P

TCCE－Pの役割は，群を構成する他のTCCE－Pと協同してDDBE－Pに対して群サービスを提供するものである。このため，次の処理を行う。

［P－　5］群開設処理
［P－　6］データ転送処理
［P－　7］群解除処理
［P－　8］ＲＥＪ処理
［P－　9］フロー制御処理
［P－１０］ＲＳＴ処理
［P－１１］ABORT処理

アボート処理以外の各処理は3.2節で記した，それぞれの手順を実現するため細部処理を付加したものである［付記図５～１１，表１～６］。また，TCCE－Pの処理は図3.3１の流れで行われる。受信処理は，ＤＬＥからフレームを受信し，そのフレームに対し必要な処理を加える（必要によりＤＬＥ又はDDBE－Pにフレームを送信する）。送信処理はDDBE－Pからフレームを受信し，そのレコードに対して必要な処理を行う（必要により，ＤＬＥ

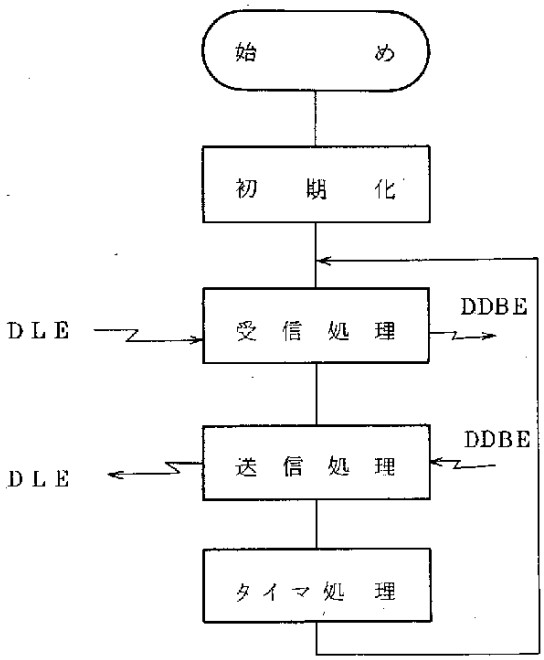


図3.3１　TCCE－Pの処理手順

又はDDBE−Pにフレームを送信する）。
以下，Ｐ－１からＰ－７について実現上の観点から付加された部分について記述する。

A. 開設処理

（ｉ）ＴＣＣＥ−Ｐは，ＴＣＥ−Ｓより起動されると，ＴＣＥ−Ｓとのインタフェースファイル ＢＣＴファイル〔図3.30〕をreadする。

（ii）ＴＣＣＥ−ＰはＢＣＴファイルのread後，ＢＣＴファイルを削除する。

（iii）(ii)の後，ＤＬＥからフレームREADYの受信を待つ。READYフレームを受信すると，実際の開設処理に移る。

（iv）開設処理は，ＳＹＮ，SYN−PAK，SYN−ACK，及びSYN−ＲＳＴの各セグメントで以下のように行う。

- ＳＹＮは，A−OPENで起動したＴＣＣＥ−Ｐ（activeな実体と呼ぶ）が送信する。
- SYN−PAKは，Ｐ−ＯＰＥＮで起動したＴＣＣＥ−Ｐ（passiveな実体と呼ぶ）がＳＹＮ受信時にＡＣＫとして送信する。
- SYN−ACKは，全実体よりＳＹＮ又はSYN−PAKを受信した時に送信する。
- SYN−ＲＳＴは，開設期群識別子（ＯＣＩＤ）の一致しないセグメントを受信した場合，開設処理間にタイムアウト等の障害が発生した場合，及び他の実体からSYN−ＲＳＴを受けた場合に送信する。

（v）全実体よりSYN−ACKを受信した場合に開設処理は終了する。この後，群の識別はＯＣＩＤから群識別子（ＣＩＤ）に移る。

（vi）ＴＣＣＥ−Ｐは開設後，DDBE−ＰにＯＰＮＤフレームを送信し，ＤＬＥにはOPND−ＣＩＤフレームでＣＩＤを通知する。

（vii）各状態で，不正な（開設期群識別子ＯＣＩＤが異なる又は受けるはずのない）セグメントを受信した場合はSYN−ＲＳＴを送信し，その後RETRYフレームでＤＬＥに開設処理の再行を依頼し，ＴＣＣＥ−Ｐ自身も初期状態のCLOSEDに戻る。ただし，ＬＩＳＴＥＮ状態での遷移はＳＹＮの受信によってのみ起る。また，状態SYN−ＳＥＮＴ時では再行回数の最大値が決めてあり（システム・ジェネレーション時に

設定），それ以上になるとTCCE−Pは終了（exit）する。

(viii) LISTEN状態を除き，各状態でタイマが動作する。タイマはその状態に遷移する時に起動され，他の状態に遷移する時に停止する。タイマが停止する前にタイムアウトになると，SYN−RSTを送信し，(vii)と同様の処理後CLOSEDに戻る。PRE−ESTABLISHED状態でタイムアウトが検出された場合には，SYN−RSTを送出するとともに，ABORTも送出する。その理由は，他のTCCE−PがESTABLISHED状態になっている可能性があるからである［付記　表1］。

B. データ転送処理

（i）データ転送処理は，データセグメント，及びACKセグメントによって行われる。データセグメントのデータ部には，DDBEからのストリームの全部又は一部が格納される。ACKセグメントは，送信処理［図3.3 1］においてDDBEからのフレームが無い場合に送出される。

（ii）送受信中，データ及びACKセグメントは，AQ1，AQ2，SQ，RTQ1，RTQ2の5つのキューで管理される。5つのキューはそれぞれ，キューコントロールブロック（QCB）を持っており，キューを個々に管理している［図3.3 2］。

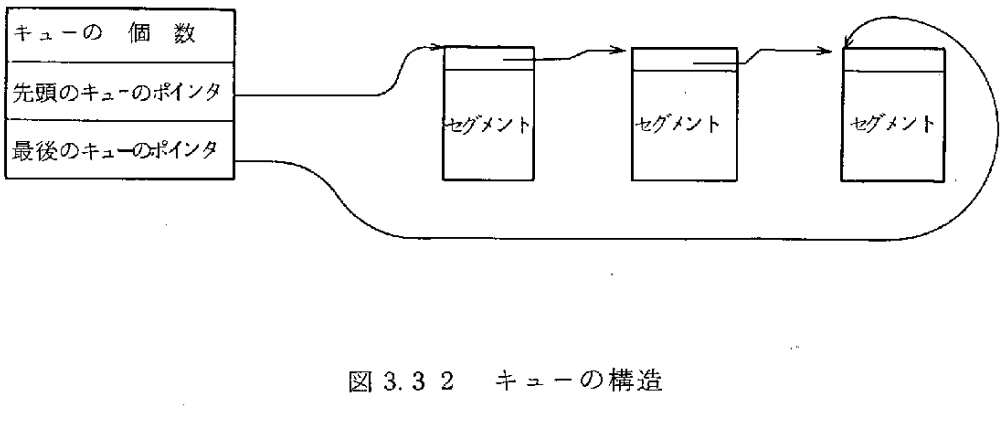


図3.3 2　キューの構造

（iii）キュー操作については，3．2．4項のデータ転送参照。

［付記　表２］

C. 解除処理

（ｉ）解除処理はＦＩＮセグメントによって行われるが，実際にはデータ転送処理の一部として行われる。

（ii）ＴＣＣＥ−Ｐは実体数個のＦＩＮフラグを持っている。ＦＩＮを受信し，ＡＣＫされた（ＡＱ２からＳＱキューへ移った）ときに，そのＦＩＮセグメントを送出した実体のＦＩＮフラグをオンとする。同様に，自実体の送出したＦＩＮがＡＣＫされた（ＲＴＱ２から落てられる）ときに，自実体のＦＩＮフラグをオンとする。これらの処理をデータ転送中に行い，全てのＦＩＮフラグがオンになるとＤＤＢＥ−ＰにＣＬＳＤフレームを送信し，ＤＬＥにはＣＬＳＤフレームを送信してＴＣＣＥ−Ｐは終了（exit）する。

D. ＲＥＪ処理

（ｉ）ＲＥＪ処理は，ＲＥＪ，ＲＥＪ−ＰＡＫ，及びＲＥＪ−ＡＣＫの各セグメントで行う。

（ii）ＲＥＪ処理は，データ転送処理中にＲＥＪ条件〔３．２．４，Ｂ参照〕が発生した場合，又データ転送処理中にＲＥＪを受信した場合に起る。

（iii）全実体よりＲＥＪ−ＡＣＫを受信した場合には，

イ）ＲＥＪ−ＡＣＫに乗っているＡＣＫリストで，ＡＣＫテーブルを更新する。

ロ）ＡＱ２のセグメントをＳＱへ移す。

ハ）ＡＱ１内でＡＣＫ（ＲＥＪ−ＡＣＫ内のＡＣＫリストの値以下のセグメント）がとれたセグメントをＳＱへ移す。

ニ）ＲＴＱ２内のセグメントを捨てる。

ホ）ＲＴＱ１内でＡＣＫがとれたセグメントを捨てる。

ヘ）ＲＴＱ１内に残ったセグメントは再送する。

［付記　表４］

E. フロー制御処理

（i）フロー制御処理は，ＲＮＲ，RNR−PAK，RNR−ACK，ＲＲ，及びRR−ACKの各セグメントで行う。

（ii）フロー制御処理は，データ転送処理（ＲＥＪ処理を含む）中に，フロー制御条件〔３．２．４，C参照〕が発生した場合，又ＲＮＲを受信した場合に起る。

（iii）フロー制御処理は，開設処理，ＲＥＪ処理と異なり，２段階から成る。第１段階は他の処理と同様に全実体からのRNR−ACK を待つまでの処理で，ここでは全実体でフロー制御処理に入った確認を行う。第２段階では，全実体がセグメントの受信が可能となった事を確認し合う。

（iv）自実体が受信不能から受信可能に成るには，ＲＮＲ処理開始後に開放したキューのサイズ（ＳＱからDDBE−Pにストリームを送信することによって開放される）を加算した合計値がある値（システム・ジェネレーション時に設定）以上になる必要がある。

（v）ＡＣＫテーブルの操作，キュー操作については，ＲＥＪ処理と同じ［付記　表５］。

F. ＲＳＴ処理

（i）ＲＳＴ処理は，ＲＳＴ，RST−PAK，及びRST−ACKの各セグメントで行う。

（ii）ＲＳＴ処理は，データ転送処理（ＲＥＪ及びフロー制御処理を含む）中に，ＲＳＴ条件〔３．２．４，D参照〕が発生した場合，又ＲＳＴを受信した場合に起る。

（iii）ＲＳＴ処理中の各セグメントヘッダのＡＣＫ部は意味を持たない（０とする）。

（iv）全点からのRST−ACK受信後は，DDBE−PにＲＳＴＤフレームを送信して，開設直後の状態に戻る。

（v）ＲＳＴ処理中は，ABORT処理以外の全ての処理に優先する。

［付記　表６］

G. ABORT処理

（ｉ）ABORT処理はABORTで行う。

（ii）ABORT処理は，DDBE−P又はＤＬＥからのABORTフレームの受信，又はABORTを受信した場合に開始される。

（iii）ABORTセグメントを受信した場合は，ABORTを送信する。ABORTの送信後，ＤＬＥにＣＬＳＤフレームを送信し，DDBE−Pには ABORTフレームを送信して，TCCE−Pは終了(exit)する。

（iv）ABORT処理は全ての処理に優先する。

（v）ABORT処理はＣＩＤが付くので，開設処理後に有効となる。

（vi）ＥＳＴＡＢＬＩＳＨＥＤ状態でABORTを受信した場合のみ，例外的に，ABORTを送出後，ＲＥＴＲＹフレームをＤＬＥに送出して，ＣＬＯＳＥＤ状態に戻る［ A. 開設処理参照 ］。

3.3.7 開発結果

現在，トランスポート群制御プロトコル（ＴＣＣＰ）を用いＬＡＮ上で単純群通信が実現されている。しかしながら，通信効率等にやや問題があり，改善の必要がある。また，ＴＣＣＰを実現するに当り理論と異なりＤＬＥ，TCCE−P，TCCE−Sの３種類の実体が必要となった。こうした問題は，使用機器等の構成が大きく影響した結果といえる。使用機器等の構成の問題として以下がある。

- Net／OneのＮＩＵのメモリー容量の不足（６４ＫＢyte）の為，ＴＣＣＰのプロセスをＮＩＵ内に実装出来ない。この結果，各ホスト内に，ＴＣＣＰプロセスを実装したために，同時に実行しているプロセスが多くなり，各ホストの処理効率が低下する。Net／OneはＬＩＳの開発には適さないといえる。
- ホスト，ＮＩＵ間のインタフェイスがＲＳ−２３２Ｃを使用しているためデータ転送速度（≦９６００bps）が遅く，Net／Oneの機能を充分活用出来ない。この為，U−１２００ではGP−IBの使用，ＳＵＮ２／１２００ではEtherne直結（１０Mbps）の機器構成を考えている。
- ＵＮＩＸは，ＴＳＳが基本となっている為，実時間処理に適さない。各プロセスが必要とするときにＣＰＵが回ってこない問題がある。例えば，群通信の多重度及び各群を構成する実体の数等が，タイムアウト値

に影響する可能性が大きい。また，ＵＮＩＸのSystem Ⅲ ではＴＣＰ／ＩＰプロトコルが使用できず，１方向通信路のパイプを用いてプロセス間通信を行う為，各種制約を受ける。オペレーティングシステムの統一の為にもU−１２００にＵＮＩＸの４．２ＢＳＤの導入が期待される。

今後，上記事項の改善を行うことがＬＩＳの処理効率の向上に直接影響を及ぼすものと考えられる。

3.4 まとめ

ＬＩＳ通信処理プロトコルの開発は，今年度実施した研究開発の主要な部分であり，その成果として，トランスポート群制御プロトコル（ＴＣＣＰ）のＵＮＩＸオペレーティングシステム上での開発が概ね終了した。また，これを利用してＬＡＮ（Ethernet）上のホスト間で，単純群通信が行なえる状態にある。現在，パフォーマンスを評価中であり，バッファのサイズ，ウインド幅，タイムアウト値等のパラメータの標準値を決めていきたい。

今後，分散型データベースシステムを取り扱う階層（ＤＤＢ層）のプロトコル（ＤＤＢプロトコル）の開発に伴い，ＤＤＢ実体との間のインタフェイスの改良及び各種サービス（エラー表示，状態の通知，緊急通信）の追加等の必要が生じる可能性がある。また，各種シュミレーションを実施し群通信サービスの効率の向上を図る必要がある。このため，トランスポート群制御プロトコルの改良を継続して実施していく予定である。

4. パソコン用データベース管理システムの実現と評価

本章では，パソコン用に開発した関係型データベース管理システム（DBMS）Granada の実現と評価について述べる。

4.1 概　要

超小型計算機の高性能化に伴い，従来大型機に構成されてきたDBSを，各個人単位に形成出来る様になってきた。この様な超小型機上のＤＢＳをパーソナルＤＢＳと呼ぶ。パーソナルＤＢＳは，外部の大型ＤＢＳからのデータ及び各個人で形成したデータを蓄積し，加工する為に用いられ，通信網で結合される事により今後の情報システムの主要な要素となっていくものと思われる。当協会では，ＬＡＮで超小型機と大型機を結合した分散型データベースシステムＬＩＳ（Local Information System ）の主要ＤＢＳとしてのパーソナルＤＢＭＳ Granadaを開発した。超小型機上に，従来のメインフレーム並の機能を有した関係ＤＢＭＳを実現することがGranadaの目標である。Granadaは，以下の機能を有している。

(1) 関係スキーマの定義・変更
(2) 述語論理形式の非手続的操作言語（ＲＱＬ）
　検索・更新（追加・消去・置換）
(3) セキュリティ管理（利用者ＩＤ，パスワード）
(4) 視野（ＶＩＥＷ）の定義・検索・更新
(5) フォーム形式の利用者インタフェース
(6) コマンドの procedure 化
(7) 物理構造の変更
(8) リカバリ機能

4.2 Granada の実現

Granadaは次の三つの階層構造から成る〔図 4.1 〕。

(1) 外部層　視野（ＶＩＥＷ）
(2) 論理層　論理データ構造とＲＱＬ演算
(3) 物理層　物理構造とアクセスパスと組単位の操作演算

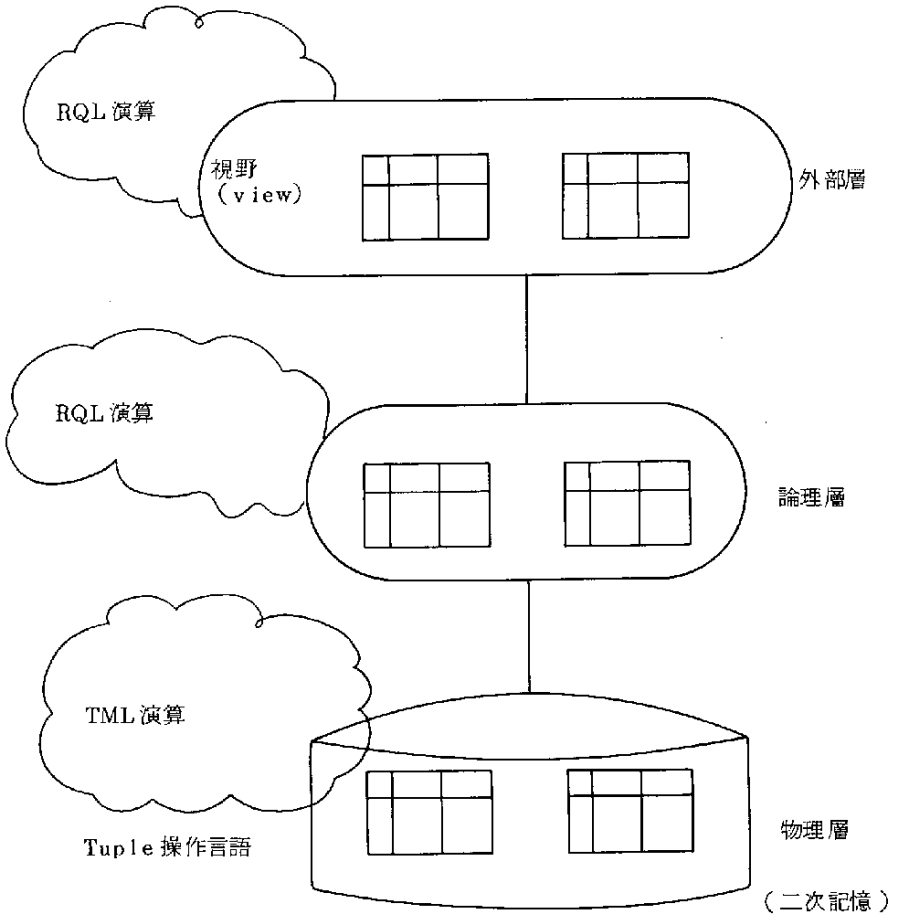


図 4.1　Granada の三層構造

4.2.1　ＤＢＭＳの論理構成

Ａ．論理データ構造

論理データ構造は，関係Ｒ（ $a_1,\cdots\cdots,a_n$ ）（ $n \geqq 1$ ）の集合から成る。Ｒの各属性には以下の特性が定義される。

$dom(a_i) = a_i$ の定義域

t_i ＝ タイプ（文字，整数，実数）

文字（１Ｂ）'ＣＮ'，文字（２Ｂ）'ＪＮ'

整数（２Ｂ）'ＩＮ'，整数（４Ｂ）'ＤＩＮ'

実数（４Ｂ）'ＲＮ'，実数（８Ｂ）'ＤＲＮ'

l_i ＝ バイト長

n_i ＝ 1 空値が許される

0 空値が許されない

i_i ＝ 1 索引を付ける

0 索引を付けない

u_i ＝ 1 重複値を許さない

0 重複値を許す

h_i ＝ 1 ハッシュ索引を付ける

0 ハッシュ索引を付けない

例

ＥＭＰ（ eno／in，ename／cn ２０，sal／in ）

属性名	タイプ	桁長	NNA	UNIQUE	INDEX
eno	ＩＮ	2	0	0	0
ename	ＣＮ	２０	0	0	0
sal	ＩＮ	2	0	0	0

Ｂ．モデュール構成

Granadaは，大きく分けて４つのモデュールから成る。(1)利用者の書類形式で検索・更新を提供するフォームインタフェース，(2)述語論理形式の非手続的演算コマンドの構文・意味解析，(3)アクセスコストの最適化，(4)物理アクセスを提供する４つのモデュールから構成される〔図4.2〕。

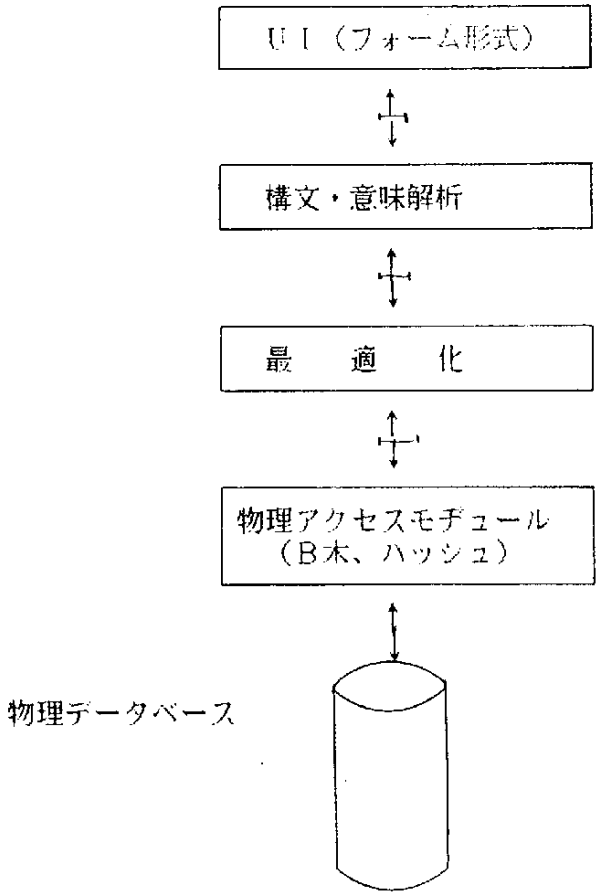


図 4.2　Granadaのモデュール構成

4.2.2　ＤＢＭＳの物理設計

二次記憶媒体は，ｎ（≧１）個の部分記憶媒体から成る。各記憶媒体は，パソコンの１つのドライヴに装備できる媒体（例，フロピィディスク，固定ディスク）に対応している。各記憶媒体は複数の関係を格納し，各関係は一つの部分記憶媒体内に格納される。各記憶媒体は，一定長（１０２４バイト）のページから成る。

関係Ｒの各組はページ内に格納され，各組はページ番号とオフセットで定まる組識別子（ＴＩＤ）を持つ。ＴＩＤは次の様に決定される。

ＴＩＤ＝ＰＡＧＥ＃　＊　１０００＋ＯＦＦＳＥＴ＃

関係の格納方法としては，(1)ある属性ａの値順（ａを整列属性とする），(2)

追加順，(3)ハッシングの３種がある。各組の文字例は，右側の空白を圧縮して格納される。

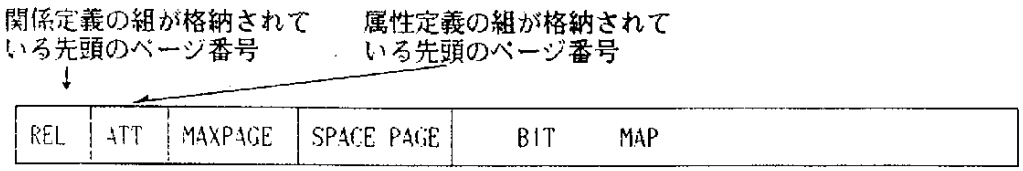
関係は，記憶媒体内で，１６ビットの関係識別子（ＲＩＤ）で表される。基本関係，導出関係に対して以下の様にＲＩＤが割り当てられる。

基本関係	ＲＩＤ（≦４０００）
REL	0
ATT	1
RTBL	2
RRTBL	3
RATBL	4
UI	5
導出関係	≧４００１

A．データベースページ管理用ページ

ページ番号０のページは二次記憶媒体のページの使用状況を管理するBit Map を格納する。Bit Map のビットがＯＮの場合そのビットに対応するページが使用されている〔図 4.3 〕。またビットがＯＦＦの場合はそのビットに対応するページが使用されていないことを示す。

この他に関係定義，属性定義が格納されているページへのポインタも格納する。



ﾍﾟｰｼﾞ 管理用ﾍﾟｰｼﾞ の構造

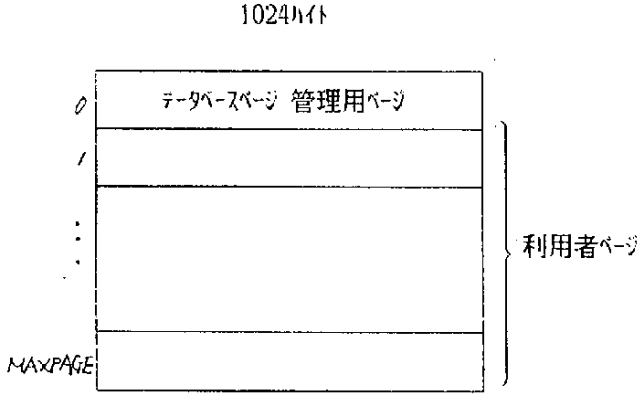


図 4.3　二次記憶の構成

B．関係の組を格納するページ構造

関係の組は物理的には，ページ単位に格納される。同一ページ内に異なる関係の組を格納することはできない。又関係の組のデータを２ページに渡って格納することも出来ない。関係の組を格納するページのフォマットは図 4.4 の様である。

F＝ページタイプ（１Ｂ）

0 ： File Page

1 ： Root Page

2 ： Non-Leaf Page

3 : Leaf Page (unique)

4 : 〃 (duplicates)

PP = RELID の前の page へのポインター (4 B)

NP = RELID の次の page へのポインター (4 B)

RELID = 関係番号(relation id) (2 B)

NT = Page 内の tuple 数 (2 B)

FP = Page 内の free page へのポインター (2 B)

$tuple_i$= 組の格納イメージ (可変長で格納)

ppt_i= i 番目の組の先頭へのポインター

図 4.4 関係の組を格納するページフォマット

各関係の組が物理的にどこのページに格納されているかは関係定義情報内に，関係の先頭の組の組識別子と最終の組の組識別子が格納される。

関係定義 (R E L)

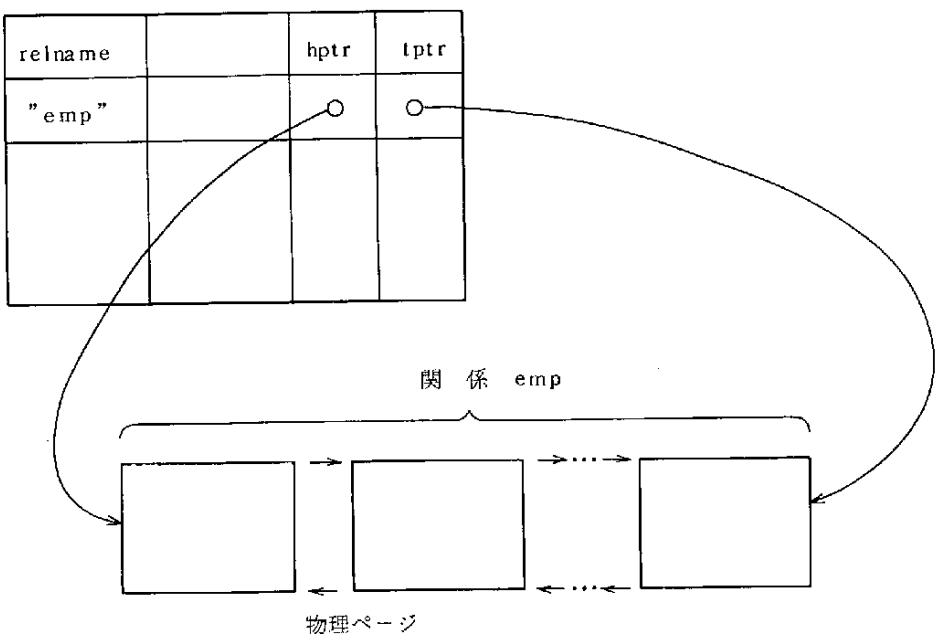
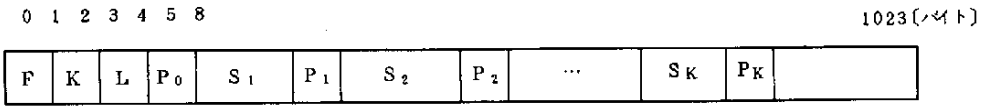


図 4.5 関係定義と関係の組を格納した物理ページとの対応

C．索引のページ構造

関係の任意の属性に対して索引を設けることが出来る。索引はB^+木で実現する。索引値のタイプが文字列の場合は右空白を圧縮してページ内に格納する。

① Root Page



F＝1：Root Page　　（1B）

　　2：Non-Leaf Page

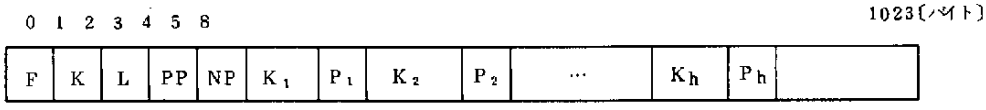
K＝Separator数　　（2B）

L＝使用バイト数　　（2B）

Pi＝下位のレベルのLeaf へのポインター（4B）

Si＝Separator

② Leaf Page(unique)



F＝3：Leaf Page　　（1B）

K＝Key数(h)　　（2B）

L＝使用バイト数　　（2B）

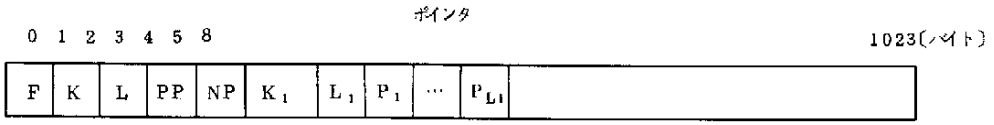
PP＝前のpage へのポインター（4B）

NP＝次のpage へのポインター（4B）

Ki＝Key

Pi＝組が格納されているページへのポインター（4B）

③　Leaf Page(duplicates)



F＝4 ： Leaf Page　　（1B）

K＝Key数　(=h)　　（2B）

L＝使用バイト数　　（2B）

PP＝前のpage へのポインター（4B）

NP＝次のpage へのポインター（4B）

Ki＝ Key

Li＝ 重複キーの数

Pi＝ 組が格納されているページへのポインター（4B）

各属性とその索引が格納されている物理ページとの対応は属性定義にB^+木のROOTページ，葉の先頭ページ，最終のページ番号を格納している。

属性定義（ATT）

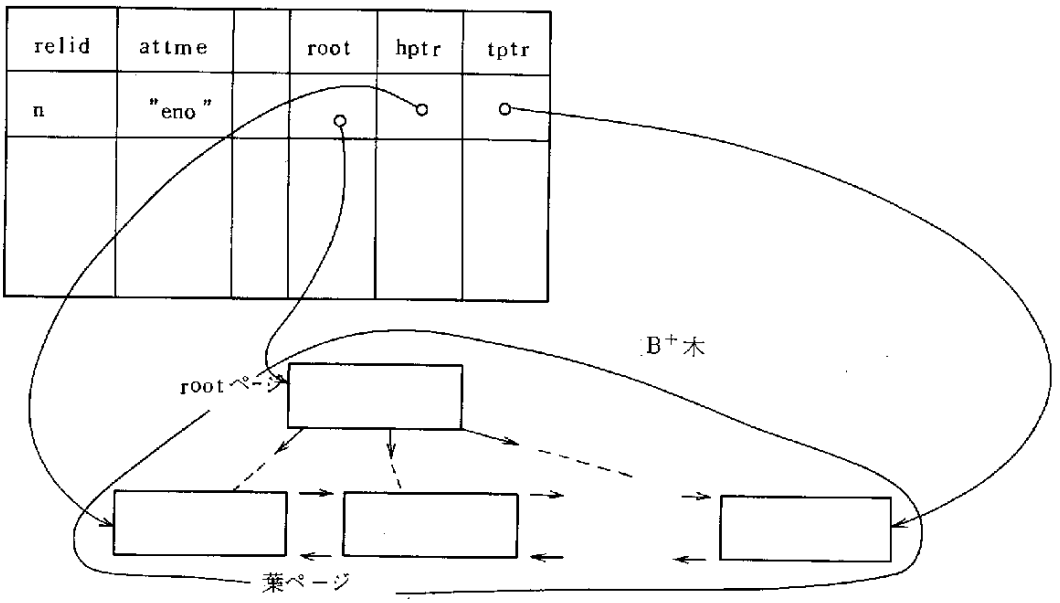


図 4.6　属性定義とB^+木との対応

4.2.3　モデュール構成

検索・更新コマンドを処理するモデュールとしては，Granadaは大きく分けて６つのモデュールから構成される〔図4.7〕。この他に検索・更新以外のコマンドを実行するモデュールが各コマンドに対して設けている。

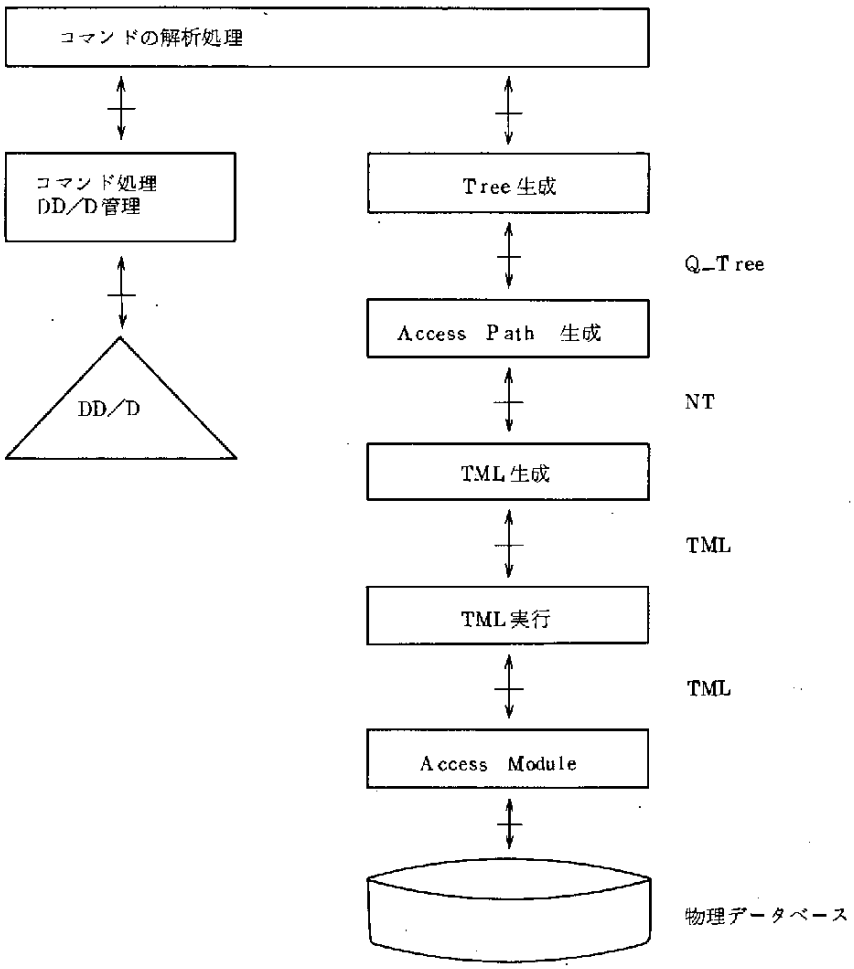


図4.7　Granadaのモデュール構成

A．ＤＤ／Ｄ（データディクショナリ／ディレクトリィ）

ＤＤ／Ｄは３つの関係ＲＥＬ，ＡＴＴ，ＵＩとから成る。それぞれの関係のスキームは表4.1の様である。ＤＤ／Ｄの関係も通常と同様にＲＱＬで操作することが出来る。

表4.1　DD／D（1）

関係定義（REL）

［バイト長］

属性名	属性番号	ﾀｲﾌﾟ	桁長	説　　明
relid	1	IN	2	関係識別子
relname	2	CN	16	関係名
degree	3	IN	2	属性数
cdate	4	CN	8	生成年月日
udate	5	CN	8	更新年月日
card	6	IN	2	組数
width	7	IN	2	組長
cls key	8	IN	2	ｸﾗｽﾀｷｰ属性番号
no key	9	IN	2	索引数
str mode	10	IN	2	格納ﾓｰﾄﾞ
owner	11	CN	6	ｵｰﾅｰ　名
h ptr	12	DIN	4	先頭の組の組識別子
t ptr	13	DIN	4	最終の組の組識別子
pcard	14	IN	2	物理ﾍﾟｰｼﾞ数
p att	15	IN	2	主属性番号
crnt	16	TID	4	組の現在子
fd	17	IN	2	ﾌｧｲﾙ識別子

IN: 整数(2B)　DIN:整数(4B)
CN: 文字　　TID:組識別子(4B)

表4.1　DD／D（2）

属性定義（ATT）

属性名	属性番号	タイプ	桁長	説明
rel_id	1	IN	2	関係識別子
att_no	2	IN	2	属性番号
att_name	3	CN	16	属性名
type	4	IN	2	属性タイプ
length	5	IN	2	属性長
offset	6	IN	2	組内のオフセット 位置
nna	7	IN	2	NULL値区分
ind_mode	8	IN	2	索引区分
unique	9	IN	2	重複区分
cls_mode	10	IN	2	クラスタ区分
hash_mode	11	IN	2	ハッシュ 区分
root_ptr	12	DIN	4	根ページ 番号
flptr	13	DIN	4	先頭の葉ページ 番号
llptr	14	DIN	4	最終の葉ページ 番号
height	15	IN	2	B_treeの高さ
card	16	IN	2	索引のカーディナリティ
sletvty	17	RN	4	選択度
ncard	18	IN	2	葉ページ 数
crnt1	19	DIN	4	現在の葉ページ 番号
crnt2	20	DIN	4	重複値の現在位置

表4.1　DD／D（3）

利用者定義（UI）

属性名	属性番号	ﾀｲﾌﾟ	桁長	説　　明
userid	1	CN	6	利用者名
passwd	2	CN	6	パスワード
role	3	IN	16	0:利用者 1: 管理者
dbname	4	CN	16	ﾃｰﾀﾍﾞｰｽ名

B．コマンド解析モデュール

データベースを検索・更新するＲＱＬ操作演算の内部表現を示す。ＲＱＬ操作演算は次の様に表現される。詳しくは4.2.5項のユーザーインタフェースを参照されたい。

(1) 検索演算

var(X_1 , X_1)…(Xn ,Xn):

get T〔$a_1=e_1$;……;$a_h=e_h$〕Q;

(2) 更新演算

追加演算

var(X_1 , X_1)…(Xn ,Xn) ;

app〈 relname 〉〔$a_1=e_1$;……;$a_h=e_h$〕Q;

消除演算

var(X_1 , X_1)…(Xn ,Xn) ;

del 〈 varname 〉Q ;

修正演算

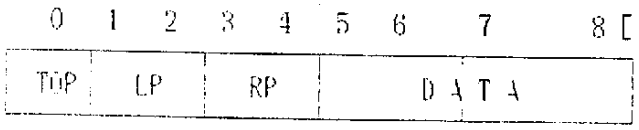
var(X_1 , X_1)…(Xn ,Xn) ;

rep 〈 varname 〉〔 $a_1=e_1$;……;$a_h=e_h$ 〕Q ;

これらのコマンドは内部的には．二進木（Q_tree）で表される。Q_treeの各ノードの構造を次に示す。

① ノード構造

基本構造




```
TOP  = [M*30+TP]*1500+OP
        M: used mark
        TP: ノードタイプ
            0: OPN      演算子ノード
            1: ATN      属性ノード
            2: VRN      変数ノード
            3: CN       文字定数ノード
            4: JN       日本語文字定数ノード
            5: IN       整数ノード
            6: DIN      倍精度整数ノード
            7: DDIN
            8: RN       実数ノード
            9: DRN      倍精度実数ノード
           10: TN       時間ノード
       OP: 演算子コード
  LP =  左ノード へのポインター
  RP =  右ノード へのポインター
  DATA= データ 領域
```

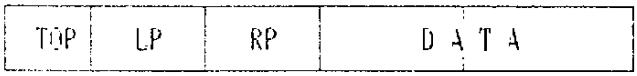
i) 定数ノード

定数ノードとしては次のものがある。

文字ノード

数ノード

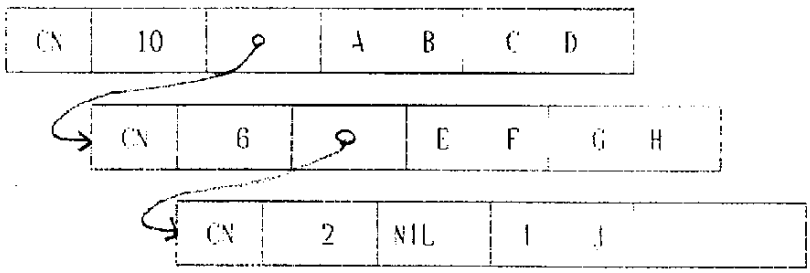
a) 文字ノード



TP:CN or IN
OP:未使用
LP:全文字数(このノード より右の文字数)
RP:次の定数ノード へのポインタ
DATA:4 文字のデータ

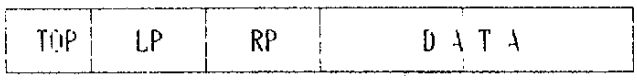
[例]

"ABCDEFGHIJ"



b) 数ノード

i)単精度整数



TP:IN
OP:未使用
LP:NIL
RP:NIL
DATA:数n を内部形式(2ﾊﾞｲﾄ)でDATAに格納する。

ロ)倍精度整数

TOP	LP	RP	D A T A

TP:DIN
OP:未使用
LP:NIL
RP:NIL
DATA:数n を内部形式(4ﾊﾞｲﾄ)でDATAに格納する。

ハ)実数

TOP	LP	RP	D A T A

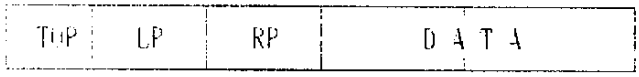
TP:RN
OP:未使用
LP:NIL
RP:NIL
DATA:数n を内部形式(4ﾊﾞｲﾄ)でDATAに格納する。

ニ)倍精度実数

TOP	LP	RP	D A T A

TP:DRN
OP:未使用
LP:倍精度実数の内部形式の頭2ﾊﾞｲﾄを格納する。
RP:倍精度実数の内部形式の頭から2ﾊﾞｲﾄ目から2ﾊﾞｲﾄを格納する。
DATA:倍精度実数の内部形式の頭から4ﾊﾞｲﾄ目から4ﾊﾞｲﾄを格納する。

ii) 属性ノード



TOP
TP:ATN
OP:属性番号
LP:RTBL#
RP:length × 100 + tpye
DATA:((((loc_mode ×10 + cluster_mode)×10
+ sort_mode)×10
+ indexed_mode)×10
+ unique_mode

loc mode	0: no hashed
	1: hashed
cluster_mode	0: non_clustered
	1: clustered
sort_mode	0: not sort_key
	1: sort_key
indexed_mode	0: non_indexed
	1: indexed
unique_mode	0: duplicate
	1: unique

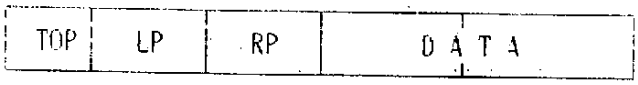
iii)演算子ノード

TP: OPN
OP: Operator Code
LP: 左の部分木へのポインタ
RP: 右の部分木へのポインタ

a)AND ・ORノード

TP: OPN
OP: ANDN(=30) ORN(=201)
LP: 左の部分木へのポインタ
RP: 右の部分木へのポインタ

b)比較演算子ノード



TP: OPN
OP: COP
LP: 左の部分木へのポインタ
RP: 右の部分木へのポインタ

c)RAT ノード

TP: OPN
OP: RATN(=32)
LP: 左の部分木へのポインタ
RP: 右の部分木へのポインタ
DATA: 結果属性番号

d)四則演算ノード

TOP	LP	RP	D A T A

TP: OPN
OP: 四則演算のoperator code
LP: 左の部分木へのポインタ
RP: 右の部分木へのポインタ

e)関数ノード

TOP	LP	RP	D A T A

TP: OPN
OP: 関数のoperator code
LP: 左の部分木へのポインタ
RP: 右の部分木へのポインタ

② get コマンドの内部表現

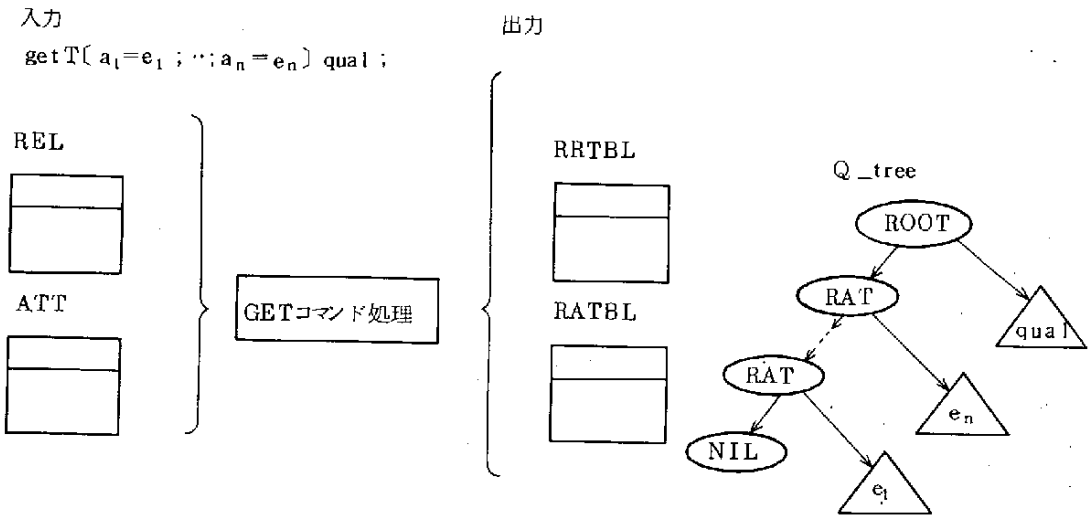


図 4.8　ＧＥＴコマンドの処理概要

③　app コマンドの内部表現

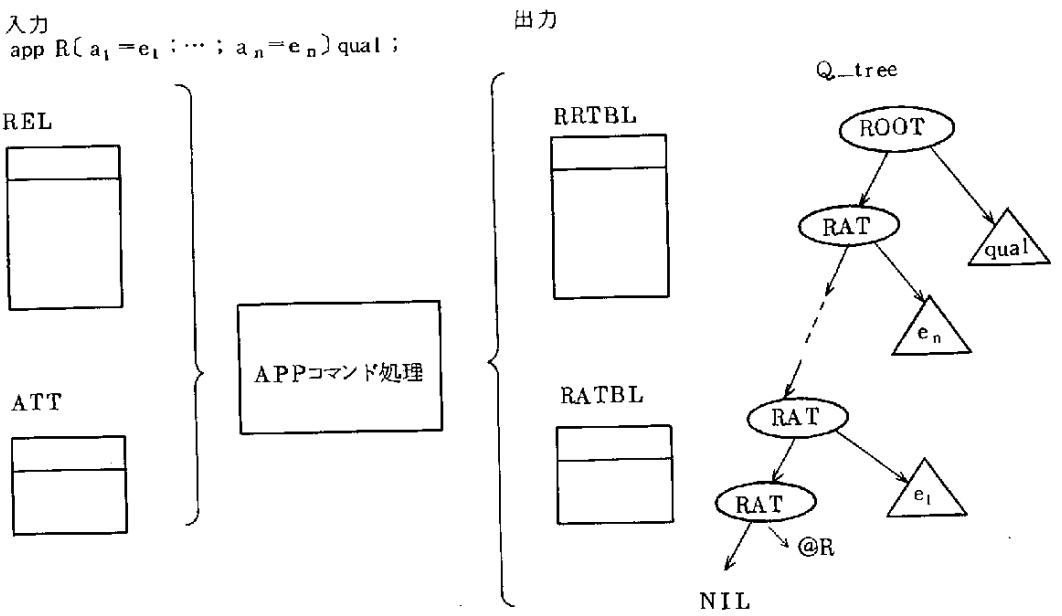


図 4.9　ＡＰＰコマンドの処理概要

④　rep コマンドの内部表現

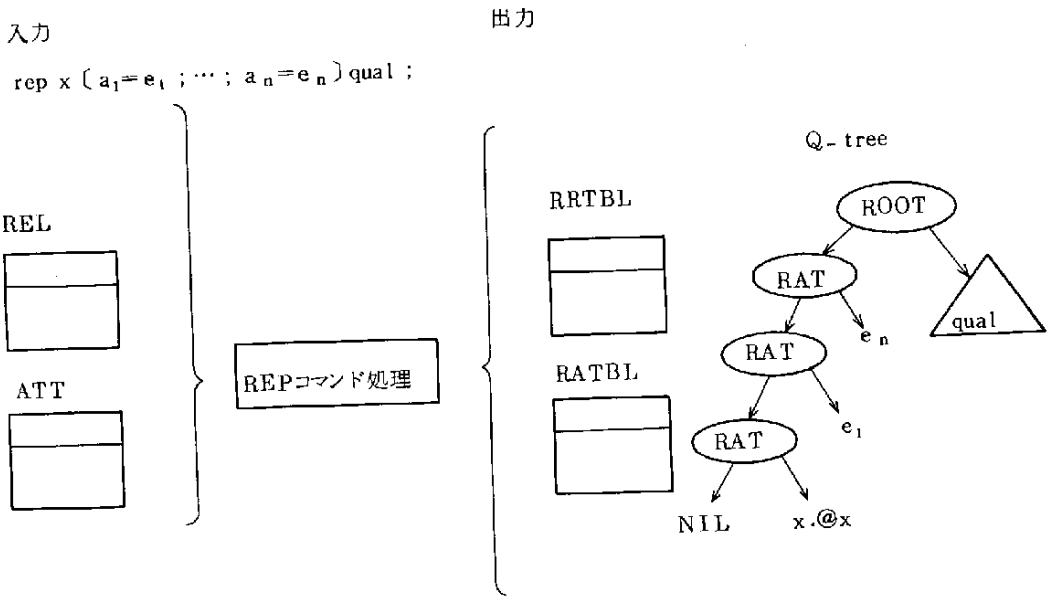


図 4.10.　ＲＥＰコマンドの処理概要

⑤　del コマンドの内部表現

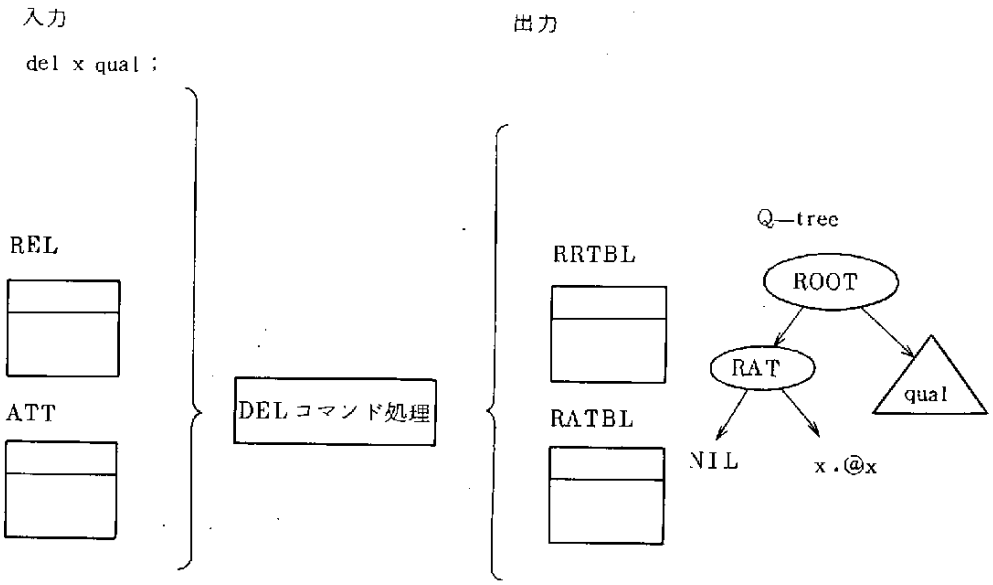


図4.11　ＤＥＬコマンドの処理概要

⑥　結果関係のＤＤ／Ｄ

関係には常駐関係と結果関係とがある。結果関係は，一つのセッション内だけ存在できる。結果関係は表4.2と表4.3に示すシステム関係（ＤＤ／Ｄ）関係により管理される。

表 4.2　ＲＲＴＢＬ

結果関係定義（ＲＲＴＢＬ）

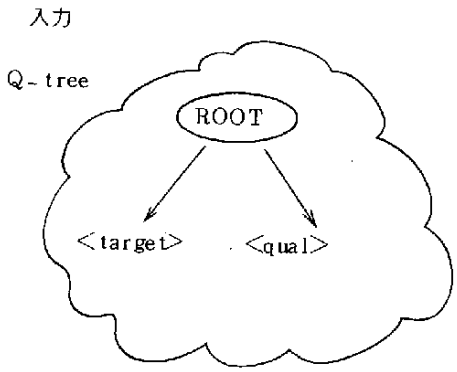
属性名	属性番号	ﾀｲﾌﾟ	桁長	説　　明
relid	1	IN	2	関係識別子
relname	2	CN	16	結果関係名
degree	3	IN	2	次数
width	4	IN	2	組長
card	5	IN	2	組数
pcard	6	IN	2	物理ﾍﾟｰｼﾞ数
uptr	7	IN	2	U_treeへのﾎﾟｲﾝﾀ
uptro	8	IN	2	U_treeへのﾎﾟｲﾝﾀ
utype	9	IN	2	問合せﾀｲﾌﾟ
BM	10	IN	2	問合せの参照する変数を表すbit_map
hptr	11	DIN	4	先頭の組の組識別子
tptr	12	DIN	4	最終の組の組識別子
crnt	13	DIN	4	現在子

表 4.3　ＲＡＴＢＬ

結果属性定義（ＲＡＴＢＬ）

属性名	属性番号	ﾀｲﾌﾟ	桁長	説　　明
relid	1	IN	2	関係識別子
attno	2	IN	2	属性番号
attname	3	CN	16	属性名
type	4	IN	2	属性ﾀｲﾌﾟ
length	5	IN	2	桁長
offset	6	IN	2	ｵﾌｾｯﾄ 値

C．ＲＱＧ生成モヂュール



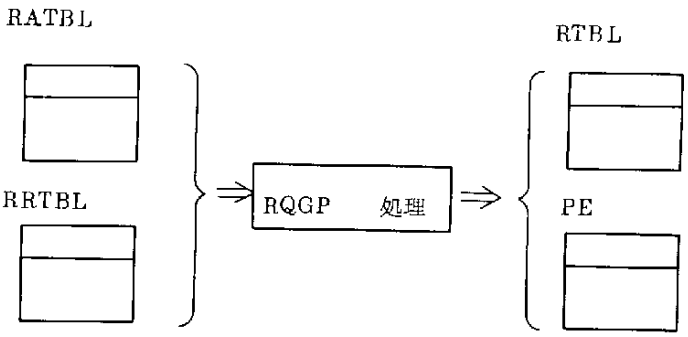


図４.１２　ＲＱＧＰの概要

ＲＱＬ問合せをグラフ表現（Relational Query Graph）に変換する。ＲＱＧは，点が組変数を，辺が結合式を，矢印が目標属性を表す。ＲＱＧは各々点と辺を表す２つの関係ＲＴＢＬ（表４.４）とＰＥ（表４.５）とによって表現される。例としては，get[d.dname;e.jname]d.dno＝e.dno;は次の様に表現される。

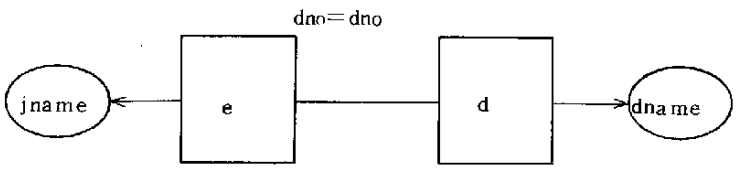


図　ＲＱＧ表現

表4.4　ＲＴＢＬ

組変数定義（ＲＴＢＬ）

属性名	属性番号	ﾀｲﾌﾟ	桁長	説　　明
var_name	1	CN	3	組変数名
relid	2	IN	2	関係識別子
qtype	3	IN	2	問合せﾀｲﾌﾟ
qptr	4	IN	2	Q_treeへのﾎﾟｲﾝﾀ
qptro	5	IN	2	Q_treeへのﾎﾟｲﾝﾀ
used	6	IN	2	作業用ﾌﾗｸﾞ
ctid	7	DIN	4	参照関係の組識別子
cbtid	8	DIN	4	索引の識別子
tlist	9	IN	2	目標ﾘｽﾄ へのﾎﾟｲﾝﾀ
rstr	10	IN	2	制限ﾘｽﾄ へのﾎﾟｲﾝﾀ
card	11	IN	2	組数
pcard	12	IN	2	物理ﾍﾟｰｼﾞ 数
pmode	13	IN	2	格納ﾓｰﾄﾞ
prstr	14	IN	2	主制限ﾘｽﾄ へのﾎﾟｲﾝﾀ
pslct	15	RN	4	prstr の選択度
nrstr	16	IN	2	非主制限ﾘｽﾄ へのﾎﾟｲﾝﾀ
nslct	17	RN	4	nrstr の選択度
nptr	18	IN	2	結合ﾘｽﾄ へのﾎﾟｲﾝﾀ
smark	19	IN	2	作業用ﾏｰｸ
mark	20	IN	2	作業用ﾏｰｸ

表 4.5　P E

結合定義（ P E ）

属性名	属性番号	ﾀｲﾌﾟ	桁長	説　　明
type	1	IN	2	ﾀｲﾌﾟ
bm	2	IN	2	Bit_map
ptr	3	IN	2	結合式へのﾎﾟｲﾝﾀ
lvar	4	IN	2	左部分木の変数番号
latt	5	IN	2	左部分木の属性番号
lpmode	6	IN	2	左部分木の格納ﾓｰﾄﾞ
lselect	7	RN	4	左部分木の選択度
lc	8	IN	2	左部分木のﾍﾟｰｼﾞ数
rvar	9	IN	2	右部分木の変数番号
ratt	10	IN	2	右部分木の属性番号
rpmode	11	IN	2	右部分木の格納ﾓｰﾄﾞ
rselect	12	RN	4	右部分木の選択度
rc	13	IN	2	右部分木のﾍﾟｰｼﾞ数
mark	14	IN	2	作業用ﾏｰｸ
mark1	15	IN	2	作業用ﾏｰｸ
lpf	16	IN	2	左属性のB木の高さ
rpf	17	IN	2	右属性のB木の高さ
uptr	18	IN	2	上方結合式へのﾎﾟｲﾝﾀ
uatt	19	IN	2	上方結合式の属性番号
datt	20	IN	2	下方結合式の属性番号
vno	21	IN	2	Bit_Mapのbiton数

D. Access Path生成モデュール

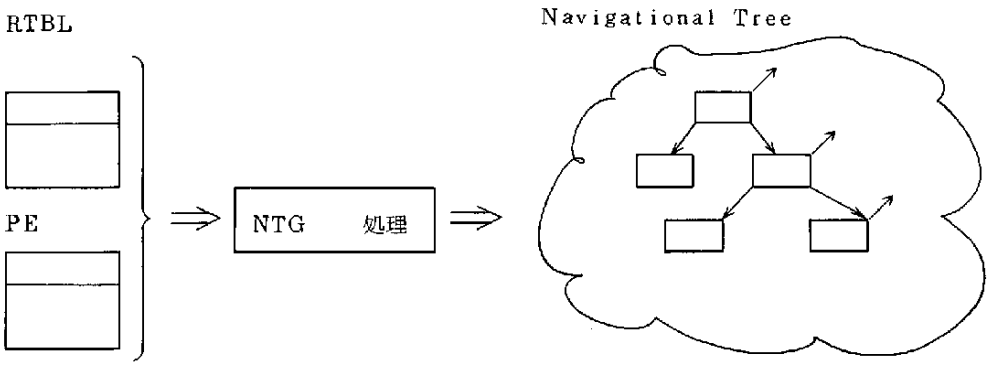


図4.13　ＮＴＧの概要

Granadaでは，ＲＱＬ検索を以下の目標値を達成する様な物理Ｉ／Ｏ手順に変換する。

①　中間結果数の最小化

②　入出力ページＩ／Ｏ回数の最小化

ＲＱＧに対して，全目標点（目標属性を持つ点）を最右の路にもつ木（巡航木）に変換する。巡航木に対して，１つの出力ファイルに対して，ある一定順序で目標集合を出力できる。

Granadaでは可能な全ケースを調べて，入出力ページ数の最小の巡航木を求めている。一回の検索で参照する組変数は，数個であると思われるので全面サーチを行っている。

E. ＴＭＬ(Tuple Manipulation Language)生成モデュール

ＴＭＬＧでは巡航木を入力して，ＴＭＬコマンドの集合としてのアクセス手順を生成する。アクセス手順として幾つかのアクセスパターンを予め設定しておく。これらのアクセスパターンのなかから，最適なアクセスパターンを見つけてＴＭＬコマンドを生成する。

以下にアクセスパターンを示す。

① 索引無しのＦＩＲＳＴアクセスパターン

```
    open
    clear
    first   < 組変数>   < 組変数> が参照している関係の先頭の組識別子を得る
    freset  < 組変数>   組識別子を現在組識別子としてRTBLに格納する
    goto L0;
    faccept < 組変数>   RTBLから現在組識別子を得る
    next    < 組変数>   現在子の次の組識別子を得る
L0: false   <DBFLAG>    組識別子が終わりかどうかを調べる
    getatt  < 組変数>   目標属性の値を読み込む
    false   <DBFLAG>>
    rstr    < 制限式>   制限式を調べる
    freset  < 組変数>   組識別子を現在組識別子としてRTBLに格納する
```

② 索引（重複無し）のＦＩＲＳＴアクセスパターン

```
    open
    clear
    key     <KEY LIST>  索引値を取り出す
    false   <KEYFLAG>   索引値が無くなったか調べる
    ifirst              この索引値を持つ組の組識別子を得る
    false   <DBFLAG >   組識別子が終わりかどうかを調べる
    getatt  < 組変数>   目標属性の値を読み込む
    rstr    < 制限式>   制限式を調べる
```

③ 索引（重複有り）のＦＩＲＳＴアクセスパターン

```
    open
    clear
    key     <KEY LIST>  索引値を取り出す
    false   <KEYFLAG>   索引値が無くなったか調べる
    ifirst              この索引値を持つ組の組識別子を得る
    reset   < 組変数>   組識別子をRTBLの現在子に格納する
    goto    L0
    reset   < 組変数>   RTBLの現在子を取り出す
    inext               現在子の次の組識別子を得る
L0: false   <DBFALG>    組識別子が終わりかどうか調べる
    getatt  < 組変数>   目標属性の値を読み込む
    rstr    < 制限式>   制限式を調べる
```

④　索引無しのＮＥＸＴアクセスパターン

```
        first   <組変数>      先頭の組識別子を得る
        freset  <組変数>      現在子としてRTBLに格納する
        goto    L0
        faccept<組変数>       RTBLから現在子を取り出す
        next    <組変数>      次の組識別子を取り出す
L0:     false   <DBFLAG>      組識別子が終わりかどうか調べる
        getatt  <組変数>      目標属性の値を読み込む
        false   <DBFLAG>
        rstr    <制限式>      制限式を調べる
        rstr    <結合式>      結合式を調べる
        freset  <組変数>      現在子としてRTBLに格納する
```

⑤　索引（重複無し）のＮＥＸＴアクセスパターン

```
        setkey    <KEY LIST>    索引値をｾｯﾄ する
        goto L0
        goto L1
        ifirst                  この索引値を持つ組識別子を取り出す
L0:     false     <KEYFLAG>     索引値の終わりを調べる
        getatt    < 組変数 >    目標属性の値を読み込む
        false     <DBFLAG>
        rstr      < 制限式 >    制限式を調べる
        rstr      < 結合式 >    結合式を調べる
```

⑥　索引（重複有り）のＮＥＸＴアクセスパターン

```
        setkey  <KEY LIST>      索引値をｾｯﾄ する
        ifirst                  この索引値をもつ組識別子を取り出す
        reset   < 組変数 >      現在子としてRTBL格納する
        goto L0
        accept  < 組変数 >      RTBLから現在子を取り出す
        setkey  <KEY LIST>      索引値をｾｯﾄ する
        inext                   次の組識別子を取り出す
L0:     false   <KEY FLAG>
        rstr    < 制限式 >      制限式を調べる
        rstr    < 結合式 >      結合式を調べる
```

Ｆ．Aggregate関数の処理方法

問合せ内のAggregate関数は，はじめに部分問合せとして処理される。単純Aggregate関数ｆは，スカラー値ｖが得られるので，問合せ内のこの関数ｆを，ｖで置き換える。一方group_by Aggregate 関数は，group_by変数をx.a,y.b……とすると，結果関係Ｇ（t,a,b……）が得られる。ｔは

Aggregate値である。Gと問合せ結果と，group_by 変数について結合をとって解を求める。

問合せに対するQ_tree から，Aggregateを処理する為にAG_treeを生成する。AG_tree から1つづAggregateを取り出して処理を行う。

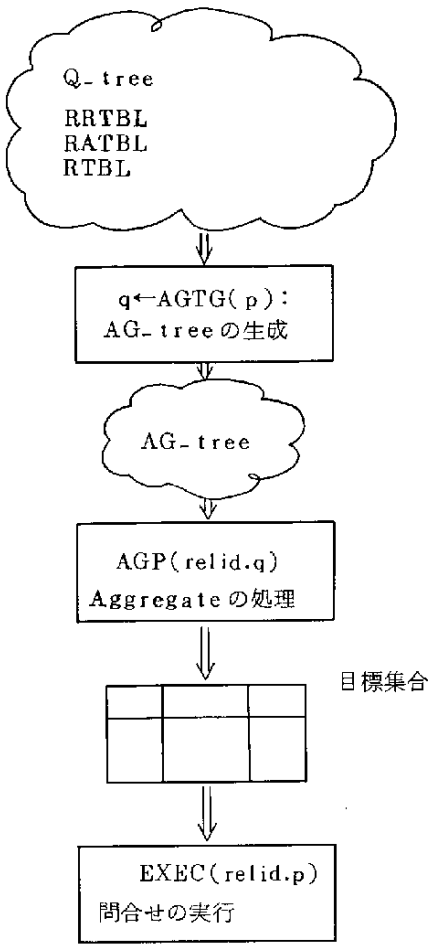


図 4.1 4　Aggregate 関数の処理概要

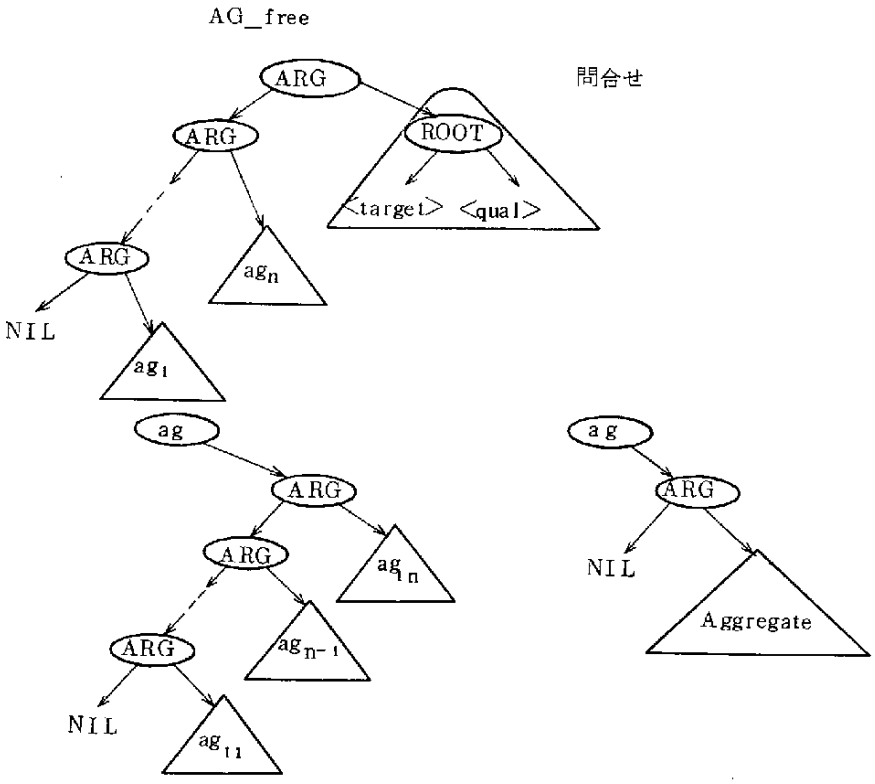
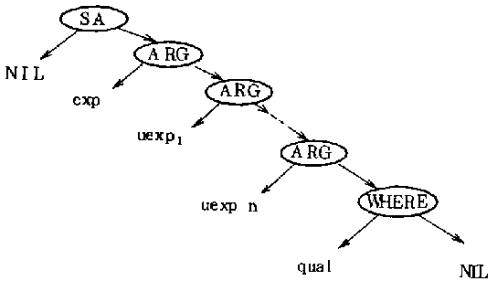


図 4.1 5　Aggregate Tree の構造

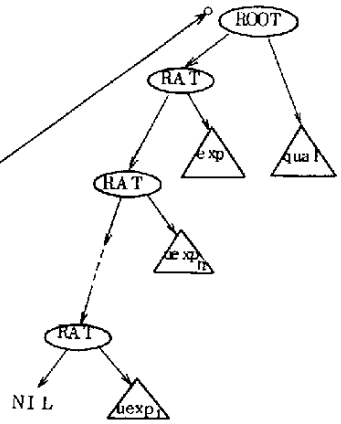
ＳＡ（Simple Aggregate）関数の処理方法



SAに対する検索木Ｑ－ tree を生成する。

RRTBL

relid		qptr
r		



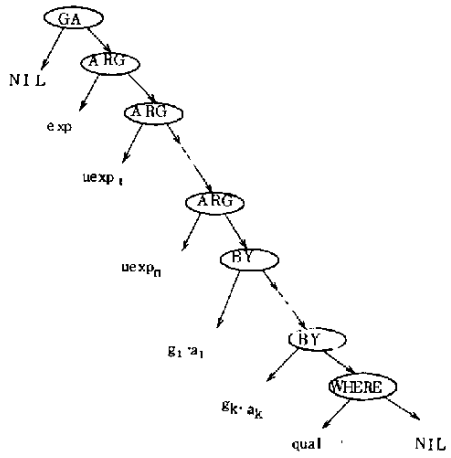
RATBL

relid	attno	---	atype	
r	1		0	uexp に対応
r	2		0	
⋮	⋮		⋮	
r	n		0	
r	n+1		2	exp

検索を実行し、目標集合を得る。目標集合に対して、Aggregate 処理を行い Aggregate 値を得る。結果を定数ノードとして格納し、参照しているＳＡノードをこの定数ノードで置換する。

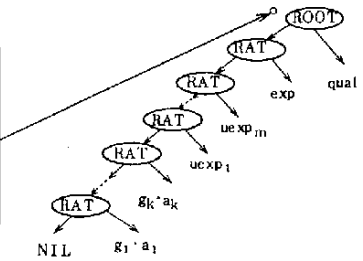
検索を実行し，目標集合を得る。目標集合に対して，Aggregate処理を行いAggregate値を得る。結果を定数ノードとして格納し，参照しているＳＡノードをこの定数ノードで置換する。

ＧＡ（Group_by Aggregate）関数の処理方法



RRTBL

relid		ptr
r		



RATBL

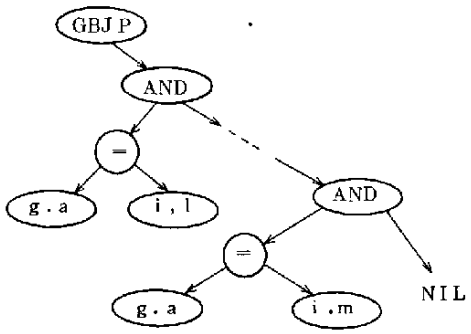
relid	attno		atyp			
r	1		1			目標集合
	⋮		⋮			g・a
r	k		1			
r	k+1		0			
	⋮		⋮			uexp
r	m(=k+n)		0			
r	m+1		2			exp

検索を実行し、目標関係を得る。目標関係に対して、Aggregate処理を行い一時関係を生成する。この時、この一時関係のGroup by 属性と元の変数の属性との結合式（ r·1=g·a and ‥‥and r·k= g·a）を生成し、これをGroup-by Join式（GBJP）とする。

検索を実行し，目標関係を得る。目標関係に対して，Aggregate処理を行い一時関係を生成する。この時，この一時関係のGroup_by 属性と元の変数の属性との結合式（ｒ.1＝g.a and…… and ｒ.k＝g.a）を生成し，これをGroup_by join式（ＧＢＪＰ）とする。

RTBL

relid	type	
r	G	



G．ＴＭＬ実行モヂュール

組単位の操作言語（Tuple Manipulation Language）を提供する。TMLとしては，表の様なコマンドがある。

ＴＭＬ実行モヂュールでは，生成されたＴＭＬコマンド列を順次読み込み，実行する。各ＴＭＬコマンドに対して，それぞれのコマンドを実行する為の関数が定義される。組単位の操作を行う為の関数としては，以下の様な関数が提供される。

表4.6　ＴＭＬコマンド

コマンド種別	コマンド名	説　　明
ファイルアクセス	first	関係の最初の組の組識別子(TID) を得る。
	next	現在TID の次の組の組識別子(TID) を得る。
	prior	現在TID の前の組の組識別子(TID) を得る。
	last	関係の最後の組の組識別子(TID) を得る。
	ifirst	同一索引値を持つ組の先頭の組の組識別子(TID) を得る。
	inext	同一索引値を持つ組の現在の組の組識別子(TID) を得る。
	ivfirst	索引の先頭値を持つ組の組識別子(TID) を得る。
	ivnext	索引の現在TID の次の組識別子(TID) を得る。
	ivprior	索引の現在TID の前の組識別子(TID) を得る。
	ivlast	索引の最後値を持つ組の組識別子(TID) を得る。
	accept	索引の現在組の組識別子(TID) を取り出す。
	reset	索引の現在組の組識別子(TID) をリセットする。
	faccept	関係の現在組の組識別子(TID) を取り出す。
	freset	関係の現在組の組識別子(TID) をリセットする。
	get	指定組識別子の組を読み込む。
出力演算	output	結果格納ｴﾘｱ のﾃﾞｰﾀ をﾃﾞｰﾀﾍﾞｰｽに書き出す。
転送演算	move	出力属性の結果格納ｴﾘｱ への転送を行う。
算術演算	add	加算を行う。
	sub	減算を行う。
	mult	掛け算を行う。
	div	割り算を行う。
算術演算	root	平方根を計算する。
	sin	ｻｲﾝ 計算をする。
	cos	ｺｻｲﾝ計算をする。
	tan	ﾀﾝｼﾞｪﾝﾄ計算をする。
	log	自然対数を計算する。
	log10	10を底とする対数を計算する。
	log2	2 を底とする対数を計算する。
	pow	べき乗を計算する。
	mod	余りを計算する。
集約演算	count	ｶｳﾝﾄを取る。
	sum	合計を取る。
	aver	平均値を取る。
	min	最小値を取る。
	max	最大値を取る。
	exist	存在を調べる。
	any	あるものを見付ける。
	median	中間値を取る。
	stev	標準偏差値を計算する。
更新演算	append	組を追加する。
	delete	組を削除する。
	replace	組を修正する。
GOTO演算	goto	次に実行するTMLｺﾏﾝﾄﾞ に跳ぶ。
比較演算	eq	等価条件を判定し、結果を判定ﾌﾗｸﾞ にｾｯﾄ する。
	ne	不等価条件を判定し、結果を判定ﾌﾗｸﾞ にｾｯﾄ する。
	gt	条件を判定し、結果を判定ﾌﾗｸﾞ にｾｯﾄ する。
	ge	〃
	lt	〃
	le	〃
真偽判定	true	判定ﾌﾗｸﾞ が真なら指定行のTMLｺﾏﾝﾄﾞ を実行する。
	false	判定ﾌﾗｸﾞ が偽なら指定行のTMLｺﾏﾝﾄﾞ を実行する。
終了ｺﾏﾝﾄﾞ	end	TMLｺﾏﾝﾄﾞ を終了する。

ＴＭＬコマンド実行関数

① 関数名

accept --- 索引の現在子を得る。

形式

```
tid=accept(a);
long  tid;          索引ページ の組識別子
struct att *a;      ATTテーブル へのポインター
```

② 関数名

reset --- 索引の現在子を ATT テーブル に格納する。

形式

```
reset(a.tid);
long  tid;          索引ページ の組識別子
struct att *a;      ATTテーブル へのポインター
```

③ 関数名

faccept --- 関係の現在子を得る。

形式

```
tid=faccept(r);
long  tid;          索引ページ の組識別子
struct rel *r;      REL テーブルへのポインター
```

④ 関数名

freset --- 関係の現在子を RELテーブルに格納する。

形式

```
freset(r,tid);
long  tid;          索引ページ の組識別子
struct rel *r;      REL テーブルへのポインター
```

⑤ 関数名

firstl --- 関係の最初の組の組識別子を得る。

形式

```
tid=firstl(r);
long  tid;          関係の最初の組識別子
struct rel *r;      REL テーブルへのポインター
```

⑥ 関数名

lastl --- 関係の最後の組の組識別子を得る。

形式

```
tid=lastl(r);
long  tid;          関係の組識別子
struct rel *r;      REL テーブルへのポインター
```

⑦ 関数名

prior1 --- 関係の現在の組識別子の1 つ前の組識別子

形式
```
tid=prior1(r);
long  tid;          関係の組識別子
struct rel *r;      REL テーブルへのポインター
```

⑧ 関数名

tid=next1-- 関係の現在の組識別子の1 つ次の組識別子

形式
```
tid=next1(r);
long  tid;          関係の組識別子
struct rel *r;      REL テーブルへのポインター
```

⑨ 関数名

first --- 組変数が参照する関係の最初の組の組識別子

形式
```
tid=first (r);
long  tid;          関係の最初の組の組識別子
struct RTBL  *r;    RTBLテーブルへのポインター
```

⑩ 関数名

last --- 組変数が参照する関係の最後の組の組識別子

形式
```
tid=last (r);
long  tid;          関係の組識別子
struct RTBL *r;     RTBLテーブルへのポインター
```

⑪ 関数名

prior --- 組変数が参照する関係の現在組の1 つ前組の組識別子

形式
```
tid=prior (r);
long  tid;          関係の組識別子
struct RTBL *r;     RTBLテーブルへのポインター
```

⑫ 関数名

next -- 組変数が参照する関係の現在組の1 つ次組の組識別子

形式
```
tid=next (r);
long  tid;          関係の組識別子
struct RTBL *r;     RTBLテーブルへのポインター
```

⑬ 関数名

get -- 関係の組を指定バッファ内に読み込む。

形式

```
get(r,tid,buf,length);
long  tid;          関係の組識別子
struct rel *r;      REL テーブルへのポインター
char *buf;          組を格納するバッファへのポインター
int  length;        組長
```

⑭ 関数名

ivfirst -- 索引値の最初の索引値へのポインター

形式

```
v=ivfirst(r,a);
struct rel *r;      REL テーブルへのポインター
struct att *a;      ATT テーブルへのポインター
char       *v;      索引値へのポインター
```

⑮ 関数名

ivnext -- 索引値の現在子の次の索引値へのポインター

形式

```
v=ivnext(r,a);
struct rel *r;      REL テーブルへのポインター
struct att *a;      ATT テーブルへのポインター
char       *v;      索引値へのポインター
```

⑯ 関数名

ivprior -- 索引値の現在子の前の索引値へのポインター

形式

```
v=ivprior(r,a);
struct rel *r;      REL テーブルへのポインター
struct att *a;      ATT テーブルへのポインター
char       *v;      索引値へのポインター
```

⑰ 関数名

ivlast -- 索引値の最後の索引値へのポインター

形式

```
v=ivlast(r,a);
struct rel *r;      REL テーブルへのポインター
struct att *a;      ATT テーブルへのポインター
char       *v;      索引値へのポインター
```

⑱ 関数名

ifirst　-- ある索引の最初の索引値を持つ組の組識別子

形式

```
tid=ifirst(r,a,v,length);
struct rel *r;      REL テーブルへのポインター
struct att *a;      ATT テーブルへのポインター
char       *v;      索引値へのポインター
int      length;    索引値の長さ
long        tid;    組識別子
```

⑲ 関数名

inext　-- ある索引の現在子の次の索引値を持つ組の組識別子

形式

```
tid=inext (r,a,v,length);
struct rel *r;      REL テーブルへのポインター
struct att *a;      ATT テーブルへのポインター
char       *v;      索引値へのポインター
int      length;    索引値の長さ
long        tid;    組識別子
```

H．アクセスモヂュール

アクセスモヂュールでは，データベースの物理ページは主記憶内にバッファ（ＭＢＦ）を設けている。

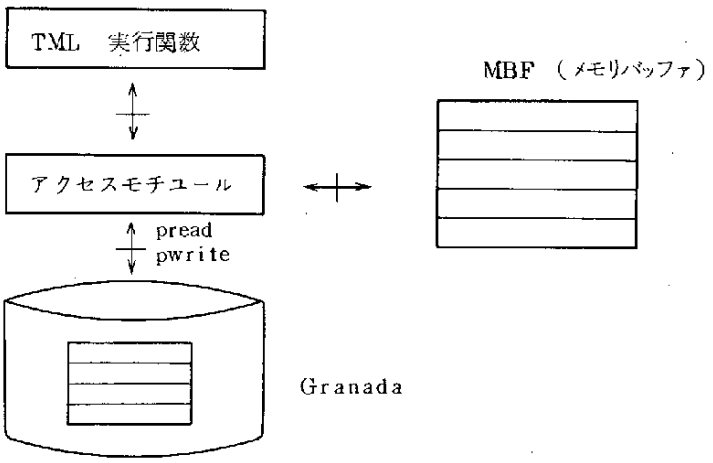


図４.１６　アクセスモヂュールの概要

データベースの物理ページに対するＩ／Ｏとしては，pread と pwrite によって実行される。このレベルでは，仮想的な一次元のページ空間に対するＩ／Ｏを行う。アクセスモヂュールは，このページＩ／Ｏを物理的な記憶媒体（drive）へのＩ／Ｏに変換する。その他にＭＢＦに物理ページを読み込んだり，書き込む為の関数等が提供される。

① 関数名

vread ----MBF に指定ページ のデータ を読み込む。

形式
```
bf=vread(p,fd);
long   p;          ページ番号
int    fd;         ファイル 識別番号
int    bf;         MBF内のバッファ番号
```

② 関数名

vwrite ---MBF 内のページ を物理的にデータベースに書き込む

形式
```
vwrite(p,fd);
long   p;          ページ番号
int    fd;         ファイル 識別番号
```

③ 関数名

vgetpage ---データベースから新しくページ を得る。

形式
```
p=vgetpage(fd);
long   p;          ページ番号
int    fd;         ファイル 識別番号
```

④ 関数名

selectbf ---MBF 内のswap outするページ を得る。

形式
```
bf=selectbf();
int    bf;         MBFバッファ番号
```

④ 関数名

updtcnt ---MBF 内のcountrをカウントダウン する。

形式
```
updtcnt(bf);
int    bf;         MBFバッファ番号
```

⑤ 関数名

writepage --データベースに指定ページ のデータ を書き込む。

形式
```
writepage(p,bf,fd);
long   p;          ページ番号
int    fd;         ファイル 識別番号
int    bf;         MBFバッファ番号
```

⑥ 関数名

readpage --ﾃｰﾀﾍﾞｰｽに指定ﾍﾟｰｼﾞ のﾃﾞｰﾀ を読み込む。

形式
```
readpage(P,bf,fd);
long   p;          ﾍﾟｰｼﾞ番号
int    fd;         ﾌｧｲﾙ 識別番号
int    bf;         MBFﾊﾞｯﾌｧ番号
```

⑦ 関数名

getpage --ﾃｰﾀﾍﾞｰｽから新しいﾍﾟｰｼﾞ を得る。

形式
```
getpage(fd);
int    fd;         ﾌｧｲﾙ 識別番号
```

⑧ 関数名

relpage --使用済みﾍﾟｰｼﾞ をﾘﾘｰｽする。

形式
```
relpage(p,fd);
int    fd;         ﾌｧｲﾙ 識別番号
long   p;          ﾍﾟｰｼﾞ番号
```

⑨ 関数名

pwrite --物理的なﾃｰﾀﾍﾞｰｽのﾍﾟｰｼﾞ の実更新を行う。

形式
```
pwrite(p,bf,fd);
int    fd;         ﾌｧｲﾙ 識別番号
long   p;          ﾍﾟｰｼﾞ番号
int    bf;         MBFﾊﾞｯﾌｧ番号
```

⑩ 関数名

pread --物理的なﾃｰﾀﾍﾞｰｽからﾍﾟｰｼﾞ を読む。

形式
```
pread(p,bf,fd);
int    fd;         ﾌｧｲﾙ 識別番号
long   p;          ﾍﾟｰｼﾞ番号
int    bf;         MBFﾊﾞｯﾌｧ番号
```

⑪ 関数名

inimag --MBF やDRVTBLの初期初理を行う。

形式
```
inimag(fd);
int    fd;         ﾌｧｲﾙ 識別番号
```

⑫ 関数名

termmag　--MBF やDRVTBLの終了初理を行う。

形式

```
termmag(fd);
int   fd;          ファイル 識別番号
```

Ⅰ．コマンド実行モデュール

① prt コマンド実行モデュール

prt コマンドでは，関係の組を順次読みだし，指定されたフォマットに従い画面に出力する。出力編集は 1 組毎に行う。編集を行う為の作業用領域として次の領域を使用する。

```
struct PLP   {
  int plp_no;              /*  行番号              */
  char plen[80];           /*  行編集エリア        */
}  plp[5];
```

又出力編集の指定情報を格納する為に次の 2 つのテーブルを使用する。

```
struct prrel   {
  int  relid;              /*  関係識別子          */
  char relname[16];        /*  関係名              */
} ;

struct pratt   {
  int  relid;              /*  関係識別子          */
  char attname[16];        /*  属性名              */
  int  attno;              /*  属性番号            */
  int  type;               /*  属性タイプ          */
  int  length;             /*  属性の桁長          */
  int  otype;              /*  出力タイプ          */
  int  olength;            /*  出力桁長            */
  int  offset;             /*  組内のオフセット値  */
} ;
```

属性値の編集

(1) 文字列の編集

文字列は右詰めとする。出力桁がデータ長よりも短い場合，出力行を増やして次の行に出力する。

出力文字列　　出力桁 20 桁

"Distributed Database Systems"

出力

Distributed Database
Systems

(2)　数字の編集

数字は左詰めとする。出力桁がデータ長よりも短い場合，出力行を増やして次の行を含めて左詰めで編集する。

出力数　　出力桁 4桁

12300

1
2300

②　分類処理モヂュール

一般に有限数列 $a_1, a_2, \cdots, a_n$ の中で $a_1 \leqq a_2 \leqq \cdots \leqq a_n$ と整順列に並んでいる部分をできるだけ長くとり，その部分を連（run)と呼ぶ。詳しくは上りの連である。図の数列では，下線を引いた部分が連で，長さ 2. 4. 1. 2 の四つの連に分割されている。5 6 2 4 8 9 3 1 7

$a_1, a_2, \cdots\cdots, a_n$ が全て相異なる数であり，n！個の可能な順列が等確率で生ずる時，連の数の期待値が（n＋1）／2であることは簡単に求まる。

二つの連がそれぞれ別のファイルに入っている，という単純な場合を考える。各連の先頭の要素を比較し，小さい方を出力し，出力した順の次の要素を入力して先頭とする。これを繰り返して，一方の連が終わりになったらば他方の残りをそのまま出力すればよい。

$$2\ 4\ 5\ 6\ 8\ 9 \Leftarrow \begin{cases} 5\ 6 \\ 2\ 4\ 8\ 9 \end{cases}$$

分類処理は大きく分けて２つのモヂュールからなる。

- 連の生成
- 連の併合

分類の入力は，分類する関係の識別番号・分類キー・キー数・分類区分

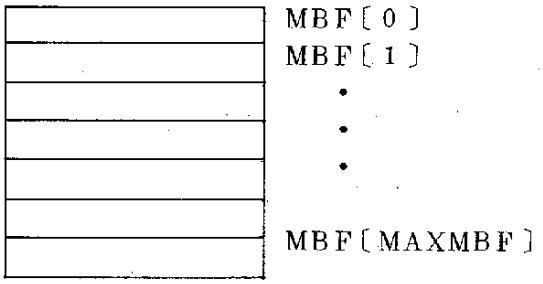
（昇順・降順）である。

(i) 連の生成処理概要

分類では，ＭＢＦ(Memory Buffer)を使用して行う。ＭＢＦはO～MAXＭＢＦまでの配列として定義されている。

(ア) ＭＢＦ〔０〕は関係からの組の読み出し用として使用する。

ＭＢＦ〔１〕～ＭＢＦ〔MAXMBF〕は分類用の作業用エリアとする。



MBF（Memory Buffer）

(イ) 関係から順にページをＭＢＦ〔０〕に読み出す。

ＭＢＦ〔０〕から，１組ずつ取り出して，順にMBF〔１〕～MBF〔MAXMBF〕に格納する。この時組ｔの（ＴＩＤ，k_1〔ｔ〕，...，k_n〔ｔ〕）を取り出して，作業用エリアに格納する。ＴＩＤは組識別子，のk_i〔ｔ〕は分類キーである。

(ウ) 作業用エリアがいっぱいになったならば，作業用エリア内の組を，分類キー順に分類する。

分類の時，組の順番の入れ替えはoffset 内のポインタの入れ替えで行う。

(エ) offsetを用いて，作業用エリア内の組を順に，二次記憶に出力する。この時，出力用のページとしてＭＢＦ〔０〕をもちいる。

(ii) 連の併合処理概要

２連の併合（binary merge）を一般化し，ｐ個のファイルにあるｐ連の併合（p_ary merge)を行うには大体次の算法となる。

算法：ｐ連の併合

1. 連の数：ｒ：　ｐを定める。
2. 以下をｒ＝１となるまで繰り返す。

2.1 残っているｒ個の連の先頭要素のうちの最小のものを出力し，最小

値の連の次の要素を読み込む。

2.2 読み込もうとした連が空になっていたならば，この事実を記憶し，rを1減らす。

3. r＝1となったならば残った連をそのまま出力して，全体を終わる。

③ テーブルの表示モヂュール

データベース内に定義されている関係の定義情報（関係名，属性名，属性のタイプetc）を表示するdspコマンドと検索結果関係の情報（結果関係名，結果属性名etc）を表示するdmpコマンドがある。

関係名はテーブル名ＲＥＬに，属性名テーブル名ＡＴＴにそれぞれ格納されている。結果関係名はテーブル名RRTBLに，結果属性名はテーブル名RATBLに，組変数と関係名との対応はテーブル名ＲＴＢＬに格納されている。

dsp …… 関係定義情報の表示（ＲＥＬ）
　　属性定義情報の表示（ＡＴＴ）

dmp …… 結果関係定義情報の表示（RRTBL）
　　結果属性定義情報の表示（RATBL）
　　組変数の表示（ＲＴＢ）
　　メモリバッファの表示（ＭＢＦ）

④ コマンド作成支援モヂュール

コマンド作成支援モヂュールには，主記憶内のコマンドバッファ(QBF)を編集するモヂュールとコマンド行を外部記憶に入出力するモヂュールがある。

i）ＱＢＦの編集モヂュール

ＱＢＦには初期状態では行番号が１０単位につけられている。この行番号を基にＱＢＦの編集（追加・削除・修正）を行う。

a．行の追加

ＱＢＦ内で行を追加する場合，追加する行間の空いている行番号を使用する。

b．行の削除

削除する行番号，又は行の範囲を指定する。

c．行の修正

修正したい行の行番号を入力し，修正したい行の内容を入力する。

d．ＱＢＦの内容表示

ＱＢＦの内容を画面に表示する。指定行または，指定範囲のＱＢＦの内容を表示する。

e．行番号の付け直し

行番号を１０から１０単位で最初の行から順に付け直す。

ⅱ）コマンドのsave／load モヂュール

コマンドを外部記憶に出力，または外部記憶から入力する。

a．外部ファイルへの書きだし

指定されたファイル名でコマンドを出力する。

b．外部ファイルからの読み出し

指定されたファイルからコマンドを読み出す。

c．不用になったコマンドをsave した外部ファイルを削除する。

⑤ 索引の生成・削除モヂュール

関係内の属性に対して，索引を生成・削除する処理モヂュールである。

ⅰ）索引の生成モヂュール

関係を順に読み出し，指定属性の値を索引に追加する。この時，指定属性が重複を許さない指定であるにもかかわらず，データ値に重複が発見された場合索引は作成されない。また既に索引がある場合，その旨を画面に表示する。

ⅱ）索引の削除モヂュール

指定された属性に索引がある場合，その索引を削除する。索引が無い場合は，その旨を画面に表示する。

⑥ 関係の生成・更新モヂュール

ⅰ）関係定義の生成モヂュール

関係定義（関係名，次数，属性構成）を生成するモヂュールである。関係の定義体は，主記憶上ではテーブル（ＲＥＬ，ＡＴＴ）に格納され，物理的にはデータベース内に１つの関係として格納される。

ⅱ）関係定義の更新モヂュール

関係定義体の名前の変更

関係名の変更

属性名の変更

関係定義体の削除

データベースから指定された関係定義体および関係の組をデータベースから削除する。

ⅲ）結果関係を関係として常駐化する。

Ⅰ．親言語インタフェース

親言語インタフェースは，利用者がデータベースをアクセスするアプリケーションを作成する為の道具を提供するものである。C言語から関数として以下の様な書式に従って呼び出される。

ⅰ）データベースの生成

関数名　crtDB

```
形式      ret = crtDB ( dbname, owner, size, passwd, dev );
    char  *dbname;      ﾃｰﾀﾍﾞｰｽ名
    char  *owner;       ｵｰﾅｰ名
    int   *size;        ﾃｰﾀﾍﾞｰｽのｻｲｽﾞ(ﾍﾟｰｼﾞ 数)
    char  *passwd;      ﾊﾟｽﾜｰﾄﾞ
    char  dev;          ﾃﾊﾞｲｽ名( 例  'c' )
    int   ret;          1   if  成功
                       -1   if  失敗
```

指定されたページ分だけデータベースをアロケート出来ない場合，取りうる最大ページ数分だけアロケートし，そのページ数をsize にセットしリターンする。

ⅱ）データベースの消去

関数名　eraseDB

```
形式      ret = eraseDB ( dbname, dev );
    char  *dbname;      ﾃｰﾀﾍﾞｰｽ名
    char  dev;          ﾃﾊﾞｲｽ名
    int   ret;          1   if   成功
                       -1   if   失敗
```

指定されたデータベースをデバイスから削除する。

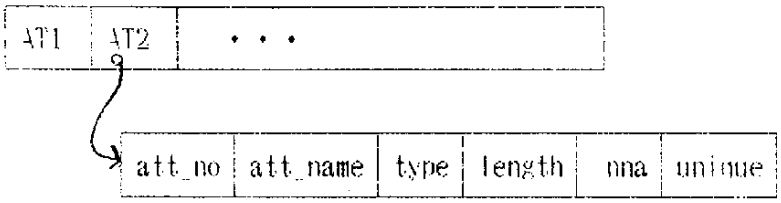
ⅲ）関係の定義

関数名　defREL;

```
形式      ret = defREL ( relname, no, att_list, dev );
    char  *relname;     関係名
    int   no;           属性数
    char  *att_list;    属性ﾘｽﾄ へのﾎﾟｲﾝﾀ
    char  dev;          ﾃﾊﾞｲｽ名
```

iii) 関係の定義

属性リスト



iv) 関係定義の削除

① 関係名による削除

関数名　dstREL

```
形式      ret = dstREL ( relname );
    char  *relname;     関係名
    int   ret;          0  if  成功
                       -1  if  失敗
```

② 関係番号による削除

関数名　dstREL1

```
形式      ret = dstREL1 ( relid );
    int   relid;        関係番号
    int   ret;           0  if  成功
                        -1  if  失敗
```

v) データベース操作

a.データベースのオープン・クローズ

①データベースのオープン

関数名　openDB

```
形式      ret = openDB ( dbname, userid, passwd, dev );
  char  *dbname;     データベース名
  char  *userid;     ユーザID
  char  *passwd;     パスワード
  char  dev;         デバイス名
```

指定のﾃﾊｲｽ上のﾃｰﾀﾍﾞｰｽ"GRANADA.DBS" をｵｰﾌﾟﾝし、useridとpasswdをﾁｪｯｸする。

②ﾃｰﾀﾍﾞｰｽのｸﾛｰｽﾞ

関数名　closeDB

形式　ret = closeDB (dev);
```
char  dev;        ﾃﾊｲｽ 名
int   ret;        0  if  成功
                  -1 if  失敗
```

指定ﾃﾊｲｽ上のﾃｰﾀﾍﾞｰｽ"GRANADA.DBS" をｸﾛｰｽﾞする。

b.組単位の操作演算

①変数定義

関数名　var

形式　v = var (relname);
```
char  *relname;       関係名
struct var *v;        構造体var へのﾎｲﾝﾀ
```

構造体var の要素をﾒﾓﾘ に確保し、その構造体へのﾎﾟｲﾝﾀをﾘﾀｰﾝする。

関数名　vari

形式　v = vari(relid);
```
int    relid;         関係番号
struct var *v;        構造体var へのﾎｲﾝﾀ
```

構造体var の要素をﾒﾓﾘ に確保し、その構造体へのﾎﾟｲﾝﾀをﾘﾀｰﾝする。

②構造体var の組ﾊﾞｯﾌｧのﾎﾟｲﾝﾀ

関数名　ctuple

形式　p = ctuple (var);
```
struct var *var;      構造体var へのﾎｲﾝﾀ
char       *p;        文字配列へのﾎｲﾝﾀ
```

構造体var の組ﾊﾞｯﾌｧの先頭ｱﾄﾞﾚｽをﾘﾀｰﾝする。

③属性値へのポインタを得る関数

関数名　　cattn

形式　　p = cattn (var, atn);
```
struct var *var;      構造体var へのポインタ
int        atn;       属性番号
char       *p;        文字配列へのポインタ
```

関数名　　catt

形式　　p = catt(var, atnn);
```
struct var *var;      構造体var へのポインタ
char       *atnn;     属性名へのポインタ
char       *p;        文字配列へのポインタ
```

④先頭組のTID を得る関数

関数名　　firstv

形式　　tid = firstv (var);
```
struct var *var;      構造体var へのポインタ
long       tid;       var で参照している関係の先頭の組のTID
```

⑤最後組のTID を得る関数

関数名　　lastv

形式　　tid = lastv (var);
```
struct var *var;      構造体var へのポインタ
long       tid;       var で参照している関係の最後の組のTID
```

⑥次の組のTID を得る関数

関数名　　nextv

形式　　tid = nextv (var);
```
struct var *var;      構造体var へのポインタ
long       tid;       var で参照している関係の次の組のTID
```

⑦前の組のTID を得る関数

関数名　　priorv

形式　　tid = priorv (var);
```
struct var *var;      構造体var へのポインタ
long       tid;       var で参照している関係の前の組のTID
```

⑧現在組のTIDを得る関数

関数名　current

形式　tid = current(var);
　struct var *var;　構造体var へのポインタ
　long　tid;　var で参照している関係の現在の組のTID

⑨組のバッファへの読み込み関数

関数名　vget

形式　vget (var);
　struct var *var;　構造体var へのポインタ

⑩属性値のバッファへの格納関数

関数名　storeatt

形式　storeatt (var, att_no, val);
　struct var *var;　構造体var へのポインタ
　int　att_no;　属性番号
　char　*val;　属性値へのポインタ

⑪組のデータベースへの追加関数

関数名　strtuple

形式　tid = strtuple (var);
　struct var *var;　構造体var へのポインタ
　long　tid;　追加された組のTID

⑫組のデータベースへの削除関数

関数名　deltuple

形式　tid = deltuple (var);
　struct var *var;　構造体var へのポインタ

c.物理構造操作演算

①索引を付ける関数

関数名　index

形式　index (rel, att);
　struct rel *rel;　構造体rel へのポインタ
　struct att *att;　構造体att へのポインタ

②索引の有無を調べる関数

関数名　chkindex

形式　ret = chkindex (relid, atn);
```
int   relid;      関係番号
int   atn;        属性番号
int   ret;        −1      if no index
                   0      if indexed
```

d.索引操作演算

①索引値を持つ先頭の組のTID を得る関数

関数名　ifrst

形式　tid = ifrst (relid, atn, val);
```
int    relid;     関係番号
int    atn;       属性番号
char   *val;      属性値へのポインタ
```

②索引値を持つ最後の組のTID を得る関数

関数名　ilst

形式　tid = ilst (relid, atn, val);
```
int    relid;     関係番号
int    atn;       属性番号
char   *val;      属性値へのポインタ
```

③現在の索引値を持つ組のTID を得る関数

関数名　icrnt

形式　tid = icrnt(relid, atn);
```
int   relid;      関係番号
int   atn;        属性番号
```

④現在の索引値を持つ組の次の組のTID を得る関数

関数名　inxt

形式　tid = inxt (relid, atn);
```
int   relid;      関係番号
int   atn;        属性番号
```

⑥現在の索引値を持つ組の前の組のTID を得る関数

```
関数名      iprior

形式        tid = iprior ( relid, atn )
    int     relid;          関係番号
    int     atn;            属性番号
```

⑥索引の最初値を持つ組のTID を得る関数

```
関数名      ilf first

形式        tid = ilf_first(relid,atn);
    int     relid;          関係番号
    int     atn;            属性番号
```

⑦索引の最後値を持つ組のTID を得る関数

```
関数名      ilf_last

形式        tid = ilf_last (relid,atn);
    int     relid;          関係番号
    int     atn;            属性番号
```

⑧索引の現在値を持つ組のTID を得る関数

```
関数名      ilf_crnt

形式        tid = ilf_crnt (relid,atn);
    int     relid;          関係番号
    int     atn;            属性番号
```

⑨索引の現在値を持つ組の次の組のTID を得る関数

```
関数名      ilf next

形式        tid = ilf_next (relid,atn);
    int     relid;          関係番号
    int     atn;            属性番号
```

⑩索引の現在値を持つ組の次の組のTID を得る関数

```
関数名      ilf_next

形式        tid = ilf_next (relid,atn);
    int     relid;          関係番号
    int     atn;            属性番号
```

⑪索引の現在値を持つ組の前の組のTIDを得る関数

関数名　　ilf_prior

形式　　tid = ilf_prior(relid,atn);
　　int　relid;　　関係番号
　　int　atn;　　属性番号

vi) 親言語インタフェースを使用した例

部(dept)に所属している従業員(emp)の番号(eno)と氏名(jname)を部毎に次のフォマットで出力するプログラムを次に示す。

出力フォマット

部番号	部名

従業員番号	氏名

〔プログラム例〕

```
#include <stdio.h>
#include <hidata1e.h>

extern char    result[];
extern long    PNIL;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/*          test prog main            */
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
main( )
{
  struct rel  *re0,   *re1,   *re2,    *srhREL();
  struct att  *at,    *srhATT();
  long   ifrst(), inxt();
  long   ilf_first(), ilf_next();
  long   tid2, tid1, tid0;
  char   dat[40], data[20], code0[4], code1[4];
  int    jp, jp0, jp1, jp2;
  int    i1, i2,  rid0, rid1, rid2;
```

```
for(i1=0;i1<4;i1++) code0[i1]=code1[i1]='¥0';

                        /*+-----------------------------+*/
                        /*|  Database Open              |*/
                        /*+-----------------------------+*/
i1=openDB("prdbs","yoko","jipdec",'c');
if( i1 ) return( -1 );

                        /*+-----------------------------+*/
                        /*| Get relation_id of "dept"   |*/
                        /*+-----------------------------+*/
if((re0=srhREL("dept"))==NULL){
    printf( "srhREL err ¥n" );
    goto 10;
}
rid0=re0->rel_id;

                        /*+-----------------------------+*/
                        /*| Get relation_id of "delnk"  |*/
                        /*+-----------------------------+*/
if((re1=srhREL("delnk"))==NULL){
    printf( "srhREL err ¥n" );
    goto 10;
}
rid1=re1->rel_id;

                        /*+-----------------------------+*/
                        /*| Get relation_id of "emp"    |*/
                        /*+-----------------------------+*/
if((re2=srhREL("emp"))==NULL){
    printf( "srhREL err ¥n" );
    goto 10;
}
rid2=re2->rel_id;


                       /*+-----------------------------+*/
                       /*| Get offset of dno in dept   |*/
                       /*+-----------------------------+*/
if((at=srhATT(rid0,"dno"))==NULL){
    printf( "srhATT err ¥n" );
    goto 10;
```

```
    jp=at->offset;

                            /*+-----------------------------+*/
                            /*| Get offset of dname in dept |*/
                            /*+-----------------------------+*/
    if((at=srhATT(rid0,"dname"))==NULL){
        printf( "srhATT err ¥n" );
        goto 10;
    }
    jp0=at->offset;

                            /*+-----------------------------+*/
                            /*| Get offset of eno in emp    |*/
                            /*+-----------------------------+*/
    if((at=srhATT(rid1,"eno"))==NULL){
        printf( "srhATT err ¥n" );
        goto 10;
    }
    jp1=at->offset;

                            /*+-----------------------------+*/
                            /*| Get offset of jname in emp  |*/
                            /*+-----------------------------+*/
    if((at=srhATT(rid2,"jname"))==NULL){
        printf( "srhATT err ¥n" );
        goto 10;
    }
    jp2=at->offset;

                            /*+-----------------------------+*/
                            /*| Check indexed or not        |*/
                            /*+-----------------------------+*/
    if(chkindx(rid0,1)==-1){
       printf( "without index ¥n" );
       goto 10;
    }

                            /*+----------------------+*/
                            /*| First read from dept |*/
                            /*+----------------------+*/
    tid0 = ilf_first( rid0, 1 );
dept:
    if( tid0 == PNIL ) goto 10;
    get( rel0, tid0, result, 0 );
                            /*+----------------------+*/
                            /*| Get data from tuple  |*/
                            /*+----------------------+*/
    i1 = getint( result, jp );                  /*dept dno*/
    strcpy( dat, &result[jp0] );                /*dept dname*/
    printf( "¥n ¥n ¥n ¥n ¥n" );
    printf( " +----+----------------------------------------+¥n" );
    printf( " |%-4.4d|%-40.40s|¥n", i1, dat );
    printf( " +----+----------------------------------------+¥n" );
    printf( "      |eno  |name                                |¥n" );
    printf( "      +-----+------------------------------------+¥n" );
```

```
    setint( code0, 0, i1 );
                                /*+--------------------------+*/
                                /*| Index first read         |*/
                                /*+--------------------------+*/
    tid1 = ifrst( rid1, 1, code0 );
delnk:
    if( tid1 == PNIL ) goto ndept;
    get( re1, tid1, result, 0 );
    i2 = getint( result, jp1 );                          /*delnk eno*/

    setint( code1, 0, i2 );
                                /*+--------------------------+*/
                                /*| Index first read         |*/
                                /*+--------------------------+*/
    tid2 = ifrst( rid2, 1, code1 );
emp:
    if( tid2 == PNIL ) goto ndelnk;
    get( re2, tid2, result, 0 );
    strcpy( data, &result[jp2] );                        /*emp jname*/
    printf( "       |%-6.6d|%-20.20s             |¥n", i2, data );
    printf( "       +-----------------------------------------+¥n" );
                                /*+--------------------------+*/
                                /*| Index next  read         |*/
                                /*+--------------------------+*/
    tid2 = inxt( rid2, 1 );
    goto emp;

ndelnk:
                                /*+--------------------------+*/
                                /*| Index first read         |*/
                                /*+--------------------------+*/
    tid1 = inxt( rid1, 1 );
    goto delnk;
ndept:
    tid0=ilf_next(rid0,1);
    goto dept;

10:
                                /*+--------------------------+*/
                                /*|  Database Close          |*/
                                /*+--------------------------+*/
    closeDB();
}
```

J．フォーム処理モヂュール

フォームとは事務処理で使用されている書類の事である。また書類の形式をフレームと呼ぶ。従ってフォームはフレームにデータを加えたものである。フォーム処理の機能としては次の２つがある。

- フレームの生成処理
- フォームの出力処理

① フレームのデータ構造

フレームの中でデータベースから読み出された属性を表示するための情報を格納するデータ構造としてＦＲＳ（Form Relation Statement）を定義する。

画面の地の部分の情報（画面の行番号と表示内容）を格納するデータ構造としてＦＢＳ（Form Backboard Statement)を定義する。

```
struct FRS   {
  unsigned int  frameid;          /* フレームの識別番号    */
  int           item;             /*                       */
  char          attname[16];      /* 属性名                */
  int           lax;              /* 左上の表示位置        */
  int           lay;              /* 左下の表示位置        */
  int           rdx;              /* 右上の表示位置        */
  int           rdy;              /* 右下の表示位置        */
  int           otype;            /* 出力タイプ            */
  int           olength;          /* 出力桁長              */
  int           fpoint;           /* 浮動小数点位置        */
  int           relid;            /* 関係識別番号          */
  int           type;             /* 属性のタイプ          */
  int           length;           /* 属性の桁長            */
  int           attloc;           /* 属性のオフセット 位置 */
} ;

struct FBS   {
  unsigned int  frameid;          /* フレームの識別番号    */
  int           line;             /* 表示行番号            */
  char          statement[80];    /* 表示内容              */
} ;
```

② フレームの生成モヂュールの概要

生成モヂュールは，ＦＢＳ，ＦＲＳに既存のデータを読み込んでから呼び出される。全く新しく作成される時は，FBS，FRSのデータはない。

またこのモヂュールが呼び出し側に復帰する時は，現在編集されたＦＢＳ，ＦＲＳを登録するか廃棄するかを判断する。

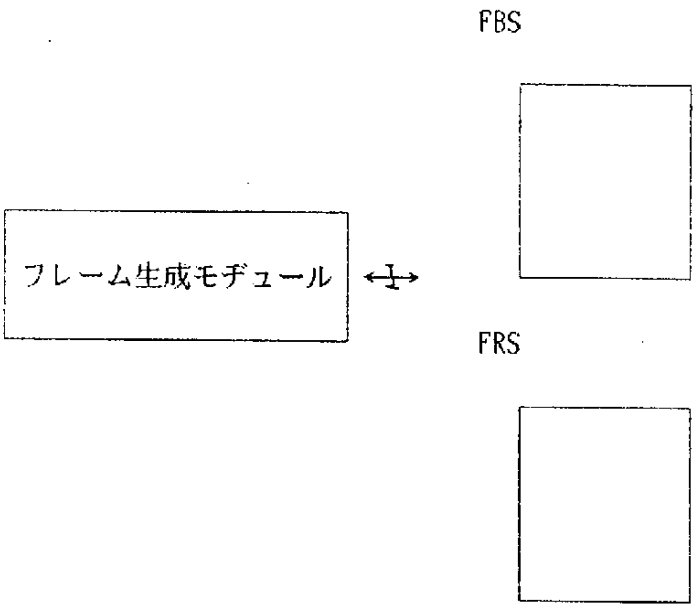


図４.１７　フレーム生成モヂュールの概要

③　フレーム出力モヂュールの概要

出力モヂュールは，タプル・データとＦＲＳ，ＦＢＳデータを与えられて呼び出される画面データ作成モヂュールと，何行目から出力するかの出力開始行データを与えられて，画面表示するモヂュールの２つにわけられるものとする。

画面データ作成モヂュールは，入力としてタプル・データ，ＦＢＳ，ＦＲＳを用いて画面データを出力するものとする。

表示モヂュールは，画面データと出力開始行データを入力として，画面データを出力するものとする。

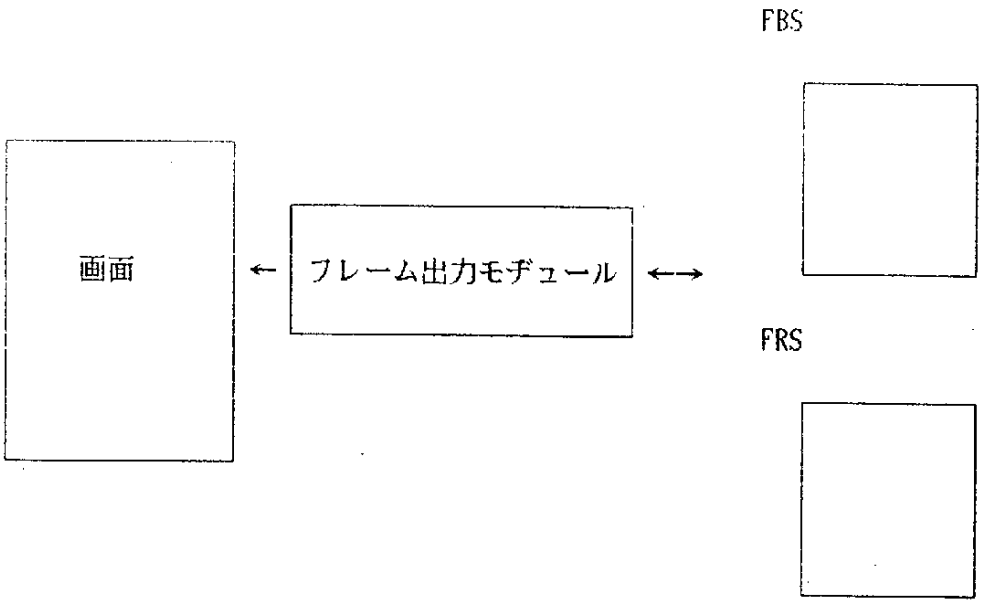
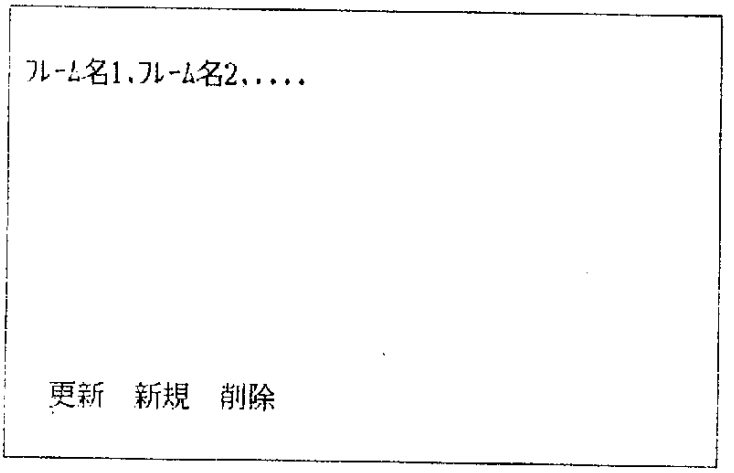


図 4.1 8　フレーム出力モデュール

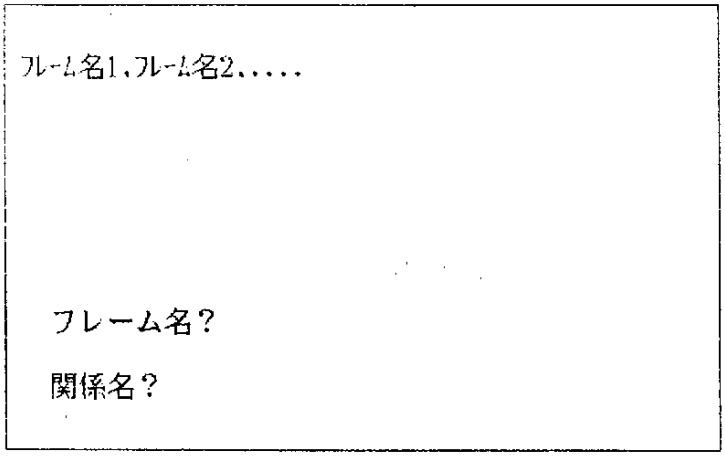
④　生成モデュールの機能

このモデュールを呼び出すと，次の様な画面が表示される。



画面 1

最初は更新がリバース表示される。カーソルを新規に移動するとそこがリバース表示される。ここで更新を選択すると次の画面となる。



画面 2

フレーム名を入力するとリテラル（ＦＢＳ，地の分）の編集モードに入る。

既存のＦＢＳ，ＦＲＳによって合成される画面を表示して，属性の編集モードに入る。

属性の編集モードでは通常のスクリーン・エディタのような動作をする。

具体的には，次の様になる。

① 矢印キーにより上下左右へのカーソル移動

② 現在のカーソル位置から文字の打ち込み

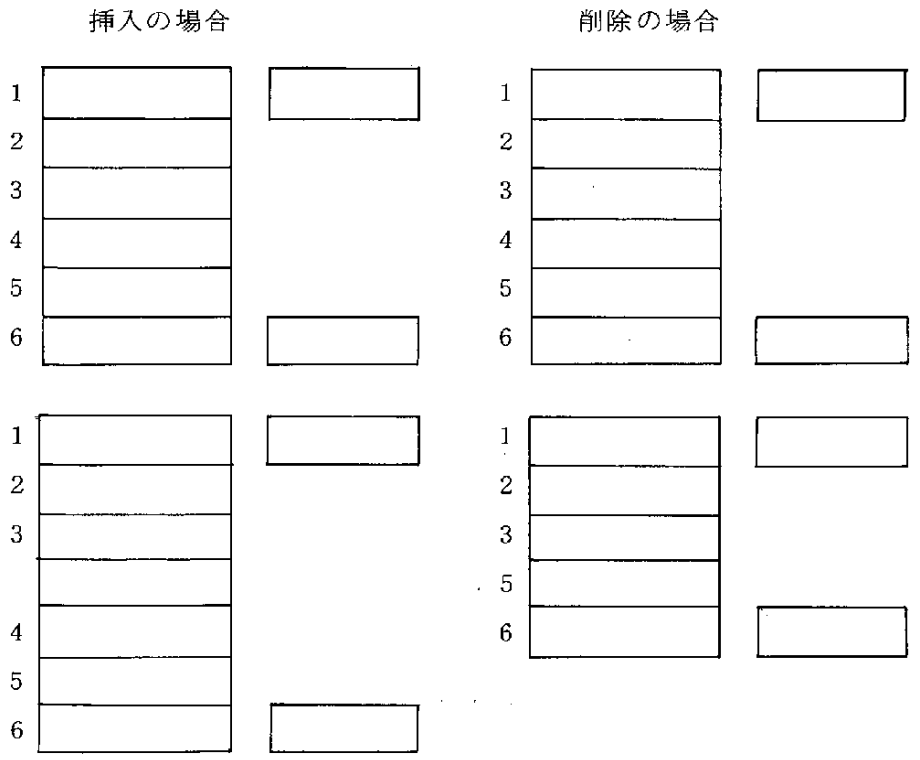
③ １行の挿入・削除

④ １行内の１文字挿入・削除

上の編集で，フレーム（ＦＲＳ）に影響を与えるのは，１行単位の挿入・削除である。

挿入して影響を受けるのは，現在のカソール行をはさんで上下にのびているフレームだけで，そフレームは挿入後１行下に延びるものとする。

削除の場合は現在のカソール行をはさんで上下に延びているフレームは下から１行上に縮み，現在のカーソル行のみに存在するフレームは消去されるものとする。



1行内の1文字単位の挿入・削除は，現在カーソルがある行の属性のみに関係するものである。

これら4つの機能のうち，挿入・削除キーはそれぞれ特殊なキーを割り当てるものとする。

属性の編集モードでないモードとして，次の2つのモードがある。

① 属性の定義・変更モード

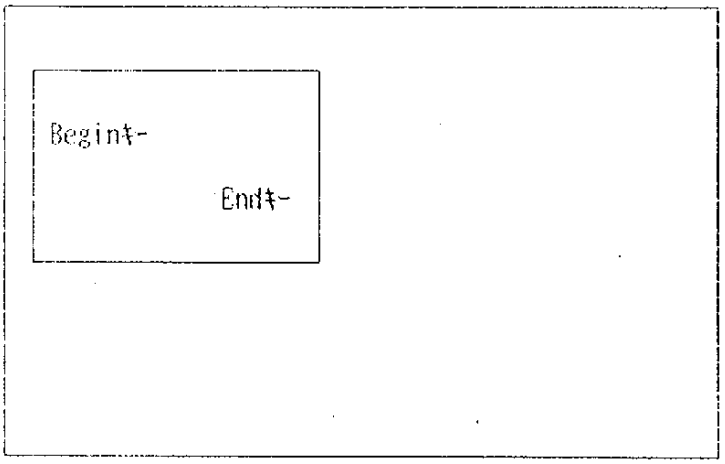
② 終了モード

これらのモードを使用する場合，プロンプトラインがとられ（通常の編集モードではプロンプトラインはない）プロンプトラインにメニューが表示される。

①の属性定義・変更モードを使用する時は，「BGEIN」キーと「END」キーによって属性を指定する。「BGEIN」キーと「END」キーで指定するエリアが，他の属性と重なった場合，エラーとする。但し，完全に重なった場合，その属性の変更・削除とみなす。

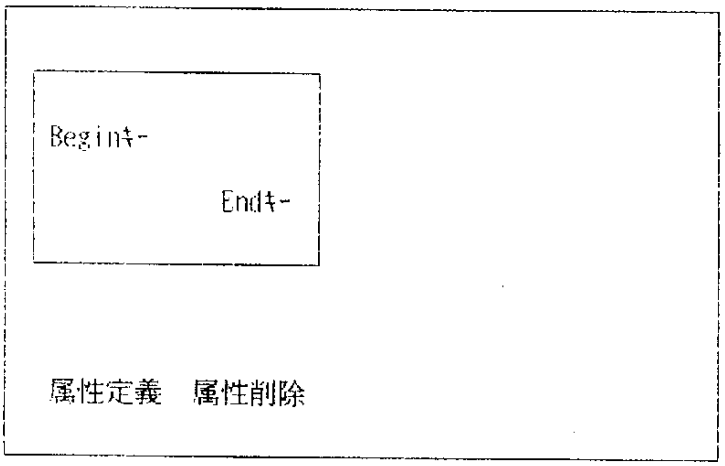
終了モードは，終了キーによって判断する。

属性の定義・変更モードで必要な入力情報は，属性のタイプ（文字データであるが数値データであるかということ）と出力フォマット（数値データの場合）である。



画面3

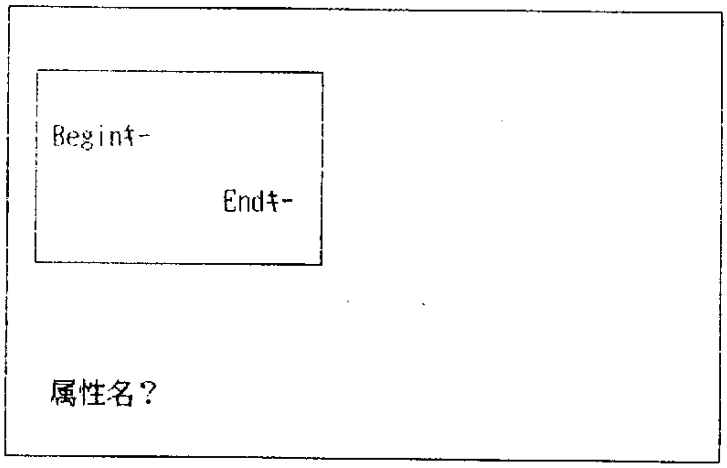
属性の定義は，最初リバースになっているものとする。この状態でRETURNキーを押すと，属性定義が選択され画面4となる。



画面4

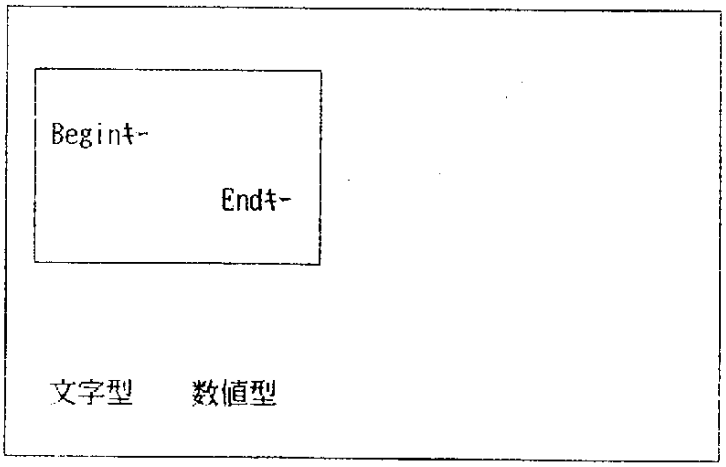
属性削除にカソールを移動すると属性削除がリバースになる。リバースになっている時にRETURNキーを押すと属性削除となる。

削除を指定したならば，属性は消去されてもとの画面に復帰し，属性定義であればつぎの表示に進むものとする。



画面 5

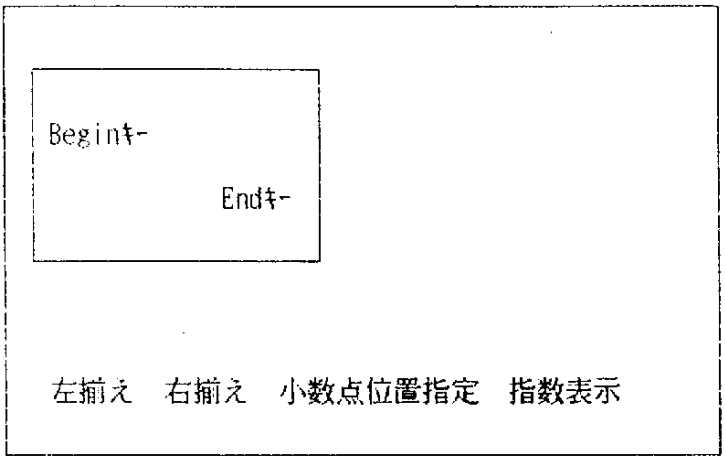
属性名を入力する画面（属性定義を選択した後に現れる画面）



画面 6

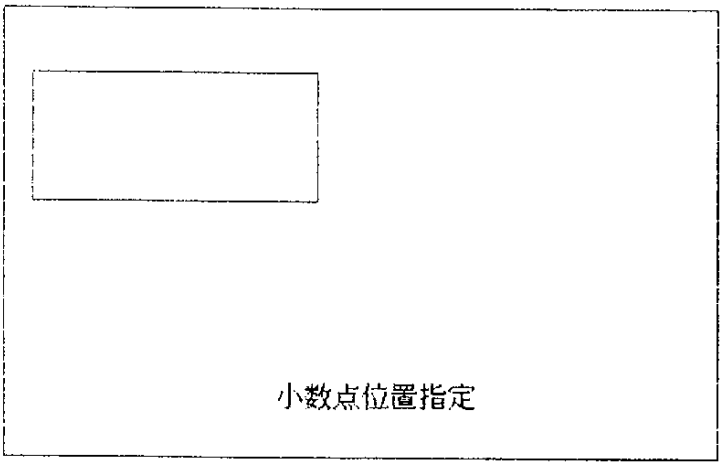
最初は図の様に文字型がリバースになっているものとする。

数値型にしたい時は，カーソル数値型に移動させるものとする。数値型にカーソルがセットされるとその部分がリバースになる。



画面 7

最初は，左揃えがリバースになっている。以下前述と同様，処理したい所にカーソルを移動させ，その項目をリバースにさせてから処理に入るものとする。

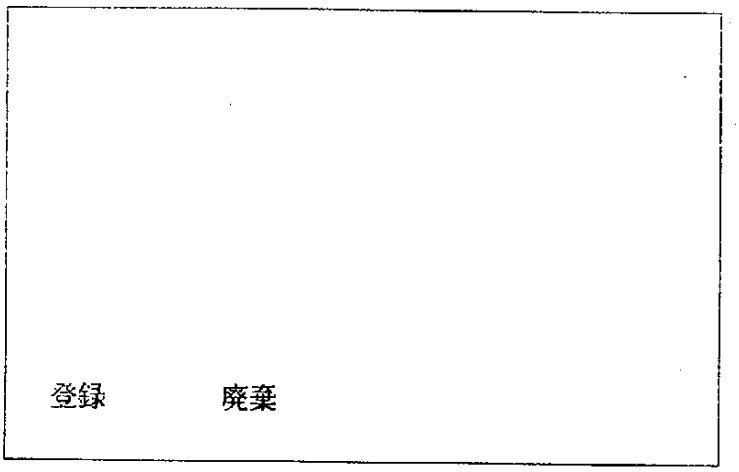


画面 7

属性のフォマットが小数点位置であった時に現れる画面。

属性部にあるカーソルを左右に移動し，”．”（ピリオド）を打つことによって位置の指示をする。

入力が終わったら，入力終了モードを指定する。入力終了モードでは，いままで編集していたフォームを登録するか廃棄するかを指定する。更新の場合の登録は，フォームの置き換えとなる。



画面 9

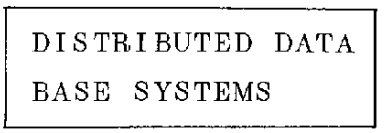
⑤ 出力モヂュールの機能

出力モヂュールは，次の２つの機能を持つ。

ｉ）画面データ編集モヂュール

ＦＢＳデータを画面データ（メモリー内バッファ）に展開し，ＦＲＳデータをもとにタプルの文字列を画面データの所定のエリアに編集する。

データ”DISTRIBUTED DATABASE SYSTEMES”



文字列データを指定された属性の中に編集する際，属性の中におさまらない文字列データは，廃棄されるものとする。

この時，１行の終わりが半角のスペースしかないにも係わらず，２バイトコードの文字がきた場合，続く行があるかないかを判定する。続く行がある場合改行して編集を続け，無い場合そこで終わるものとする。

数値データに関しては，左揃え・右揃え・小数点位置指定のフォーマットに従い，次の様に表示される。

左揃え	右揃え	小数点位置指定
123	123	123.00

数値データがオーバーフローした場合，（”ＸＸ．．Ｘ”）と表示する。

全ての画面データ編集が終わったならば，呼び出し側に復帰する。

ⅱ）画面データの表示モヂュール

画面データを，ＣＲＴ画面に指定された行から２３行表示して復帰する。

⑥　画面制御文字

画面制御文字の機能

^T	ラインドップ	
^I	TAB	
TAB	TAB	
^H	←	カーソル移動
^J	↓	
^K	↑	
^L	→	
^Y	ラインデリート	
^U	ラインインサート	
DEL	文字デリート	
INS	文字インサート	
^@	再表示	
^B	項目の最初	
^E	項目の終わり	
^Q	QUT	

4.2.4　開発環境

Granada Ver1.0 はＣ言語を開発言語として採用した。現状では，超小型

機で大きなシステムを開発するには，モヂュール単位にcompileの出来るC言語が適切である。又他機種への移値を考えてもC言語で開発すると移植が容易である。開発用の超小型機としては，日電のPC9801（機器構成図4.19）のMS－DOSの下に応用プログラムとして作成した。OSとしてMS－DOSを使用したのは，UNIX-likeのファイル管理およびHard DISKの使用が可能であることがその主な理由である。

C言語としては，Lattice Cを使用して開発を行った。Lattice Cはメモリを1MBまで使用することが可能であり，大きなシステムを作成する事が容易である。ソースプログラムの入力はフルスクリーンエディタPMATEを使用した。開発したシステムのソースプログラムのステップ数は約20,000行である。

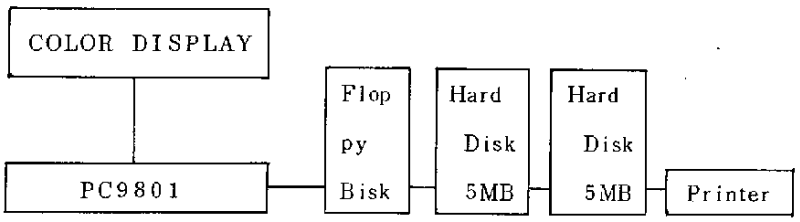


図4.19　PC9801の機器構成

現在は，SUN2／120（UNIX 4.2BSD）に移植し，ソフト開発はSUNで行っている。

4.2.5　ユーザインタフェース（コマンド仕様）

A．管理者用インタフェース

管理者用には，データベース内に利用者が使用する関数及び属性の定義を行うコマンドがある。

データベースを作成するには，まづ最初にデータベースの領域を二次記憶に取る必要がある。データベースの領域が取られたら，利用者が利用する関係の定義をデータベース内に作成することが必要である。関係の定義を作成する為のコマンドとしてはcrtコマンドがある。

① crtコマンド

crt〈関係名〉(〈属性名〉／〈タイプ〉[〈桁長〉]{，〈属性名〉／

〈タイプ〉〔〈桁長〉〕｝）；

〈タイプ〉：：＝CN ｜ JN ｜ IN ｜DIN ｜ RN ｜ DRN

属性のタイプが文字タイプ（即ち，ＣＮ，ＪＮの場合）は，桁長を省略出来ない。crtコマンドで生成された関係・属性定義は，それぞれ関係テーブル（ＲＥＬ），属性テーブル（ＡＴＴ）に格納される。

〔例〕

```
crt emp (eno/in, efname/cn20, ename/cn20, jname/jn20,
         esex/cn1,byear/in,hyear/in,position/jn10,ext/in);
```

REL

relid	rel_name	degree	width	
20	emp	9	79	

ATT

relid	att_no	att_name	type	length	
20	0	@emp	DIN	4	
20	1	eno	IN	2	
20	2	efname	CN	20	
20	3	ename	CN	20	
20	4	jname	JN	20	
20	5	esex	CN	1	
20	6	hyear	IN	2	
20	7	byear	IN	2	
20	8	position	JN	10	
20	9	ext	IN	2	

```
crt dept (dno/in, dname/jn50);
```

REL

relid	rel_name	degree	width	
21	dept	2		

ATT

relid	att_no	att_name	type	length	
21	0	@dept	DIN	4	
21	1	dno	IN	2	
21	2	dname	JN	50	

crt pelnk (pno/cn8, eno/in) ;
REL

relid	rel_name	degree	width	
22	pelnk	2		

ATT

relid	att_no	att_name	type	length	
22	0	@pelnk	DIN	4	
22	1	pno	CN	8	
22	2	eno	IN	2	

② rnmコマンド

rnm〈関係名〉by 〈関係名'〉;

関係名を変更したい時に使用するコマンドである。但し，新しい関係名と同じ名前が既にデータベース内にある場合は変更出来ない。

〔例〕

関係名 empを employee に変更する為のコマンドは次の様である。

```
rnm emp by employee;
```

関係名 proj を project に変更する為のコマンドは次の様である。

```
rnm proj by project;
```

③ anmコマンド

anm〈属性名〉by 〈属性名'〉on〈関係名〉;

属性名を変更したい時に使用するコマンドである。但し，新しい属性名と同じ名前がすでに関係内にある場合は変更出来ない。

〔例〕

関係 emp内の属性名 efname を fmilyme に変更する為のコマンドは次の様である。

anm efname by fmilyme on emp;

④ modコマンド

mod＜結果関係名＞;

結果関係名を常駐関係としてデータベースに登録する為に使用される。結果関係はセッションが終了するとデータベースから消去されるが，常駐関係は消されないで保存される。

[例]
結果関係名testを常駐関係とてデータベースに定義する為のコマンドは次の様である。
mod test;

⑤dst コマンド

dst <関係名>;

常駐関係をデータベースから消去するコマンドである。関係の組もデータベースから消去される。

[例]
データベースから関係名emp を消去する為のコマンドは次の様である。
dst emp;
このまま作業をつづけますか。
実行する = y 実行しない = n

処理をキャンセルする場合はここで 'n'を入力する。

データベースから関係名projを消去する為のコマンドは次の様である。
dst proj;

⑥ind コマンド

ind <関係名> on <属性名>;

関係の属性に索引を付ける為のコマンドである。

[例]
関係emp 内の属性eno に索引を付ける為のコマンドは次の様である。
ind emp on eno;

関係emp 内の属性efnameに索引を付ける為のコマンドは次の様である。
ind emp on efname;

⑦din コマンド

din < 属性名> in <関係名>;

関係に付けられた索引を消去する為のコマンドである。

[例]
関係emp 内の属性eno に付けられていた索引を消去する為のコマンドは次の様である。

```
din eno in emp;
```

関係emp 内の属性efnameに付けられていた索引を消去する為のコマンドは次の様である。

```
din efname in emp;
```

B. 利用者インタフェース

利用者に対してデータベースの検索・更新（追加，削除，修正）用のコマンドとその実行コマンドがある。

①var コマンド

var (<変数名>.< 関係名>)　{(<変数名>.< 関係名>)} ;

検索・更新で参照する関係に対する変数を宣言する。< 変数名> は、頭1 文字は英字で始まる2 文字までの英数字で定義する。

[例]
関係emp に組変数名としてe を定義するコマンドは次の様である。

```
var ( e. emp );
```

関係projに組変数名としてp を定義するコマンドは次の様である。

```
var ( p. proj );
```

②get コマンド

データベースの検索を定義するコマンドである。

```
get {< 結果関係名> } [ {< 属性名>=} < 項 > {, {< 属性名>=} < 項 >} ]
                       {< 条件式 >} ;
< 項 >::= < 定数 > | < 変数名 >.<属性名> | < 項> の算術式
< 条件式 >::= < 項 > の比較式 | < 項 > の比較式 and < 項 > の比較式 |
                          < 項 > の比較式 or < 項 > の比較式
< 算術関数 >::= + | - | * | / | pow | sin | cos | tan | sqrt | mod |
                log
< 集約関数 >::= min | max | sum | aver | count | exist | any | median |
                sdev
< 比較演算子 >::= = | > | < | >= | <= | <>
```

[例]

①関係emp の全組を検索するコマンドは次の様である。

```
var ( e. emp );
get [ e.all ] ;
```

②上の問合せに従業員番号(eno) が100 以上の組の従業員番号と氏名(jname) を求めるコマンドはつぎの様である。

```
var ( e, emp );
get [ e.eno; e.jname ] e.eno >= 100;
```

③上の問合せに性名(efname)が"takizawa"であるものという条件を付ける。

```
var ( e, emp );
    get [ e.eno; e.jname ] e.eno >= 100 and e.efname = "takizawa";
```

④算術演算の例

```
get [ add    = 20 + 10];
get [ minus  = 20 - 10];
get [ multpl = 20 * 10];
get [ div    = 20 / 10];
get [ pow    = pow ( 2, 2 )];
get [ sin    = sin ( 0.1 ) ];
get [ cos    = cos ( 0.1 ) ];
get [ tan    = tan ( 1.0 ) ];
```

⑤集積関数の例

```
var (d,dept) (e,emp);
```

部番号(dno) の最小値を求める問合せは次の様である。

```
get [ min = min ( d.dno )];
```

部番号(dno) の最小値を求める問合せは次の様である。

```
get [ max = max ( d.dno )];
```

部番号(dno) の合計値を求める問合せは次の様である。

```
get [ sum = sum ( d.dno )];
```

部番号(dno) の数を求める問合せは次の様である。

```
get [ count = count ( d.dno ) ;
```

従業員の年齢の平均値を求める問合せは次の様である。(ユニーク変数eno を用いる)

```
get [ aver  = aver ( 85 - e.byear , e.eno ) ];
```

従業員番号(eno) の中間値を求める問合せは次の様である。

```
get [ median = median ( e.eno )];
```

部番号(dno) が40以上の部のある1 つの部名を求める問合せは次の様である。

```
get [ any = any ( d.dname ) ] d.dno > 40;
```

部番号(dno) が60より大きい部が存在するかどうかを求める問合せは次の様である。存在する場合は1 がリターンされ、存在しない場合は0 がリターンされる。

```
get [ exist = exist ( d.dno where d.dno > 60 )];
```

Group by集積関数の例

```
var ( d, dept ) ( de, delnk );
```

部に所属している従業員の中で最小の従業員番号を部毎に求める問合せは次の様である。

```
get [ d.dname; min=min( de.eno by d.dno  where d.dno = de.dno )];
```

部に所属している従業員の中で最大の従業員番号を部毎に求める問合せは次の様である。

```
get [ d.dname; max=max( de.eno by d.dno where d.dno = de.dno )];
```

部に所属している従業員の中で平均の従業員番号を部毎に求める問合せは次の様である。

```
get [ d.dname; avr=aver(  de.eno by d.dno where d.dno = de.dno )];
```

部に所属している従業員の数を部毎に求める問合せは次の様である。

```
get [ d.dname; mem=count (  de.eno by d.dno where d.dno = de.dno )];
```

部に所属している従業員の従業員番号の合計を部毎に求める問合せは次の様である。

```
get [ d.dname; sum=sum (  de.eno by d.dno where d.dno = de.dno )];
```

③app コマンド

データベース内の関係に組を追加するコマンドである。

```
app < 関係名 > [ <属性名> = < 項 > {、 <属性名> = < 項 >  } ]
               { <条件式 >}  ;
```

[例]

関係deptに組(dno=90,dname=" 基礎ｼｽﾃﾑ研究開発室")を追加するｺﾏﾝﾄは次の様である。

```
app dept [ dno =90; dname=" 基礎ｼｽﾃﾑ研究開発室"];
```

関係emp に組(eno=272,jname="横塚　実",efname="yokotsuka",ename="minoru")を追加するｺﾏﾝﾄは次の様である。

```
app emp [ eno=272; jname="横塚　実";efname="yokotsuka";ename="minoru"];
```

④del コマンド

データベース内の関係の組を消去するコマンドである。

```
del < 組変数名 >        { <条件式 >}  ;
```

[例]

関係deptの組を全て消去するｺﾏﾝﾄは次の様である。

```
var ( d, dept );
del d;
```

関係deptの組で部番号が20以上の組を消去するｺﾏﾝﾄは次の様である。

```
var ( d, dept );
del  d d.dno >= 20;
```

⑤rep コマンド

データベース内の関係の組を修正するコマンドである。

```
rep < 組変数名 > [ <属性名> = < 項 > {、 <属性名> = < 項 >  } ]
               { <条件式 >}  ;
```

[例]

関係deptの組で部名(dname) が "管理課" である組の部名を "電子計算機運用管理室" に変更するｺﾏﾝﾄは次の様である。

```
var ( d, dname );
rep d [ dname =  "電子計算機運用管理室"] d.dname = "管理課";
```

⑥srt コマンド

関係の組を指定属性の値で分類する。

srt < 関係名 > {on <属性名> {, < 属性名> } } {in a d }

{to <結果関係名> } ;

a=ascending (昇順)　d=descending (降順)

[例]
関係emp を属性efnameで降順で分類し、結果を結果関係名testに格納する為のコマンドは次の様である。

```
srt emp on efname in d to test;
```

関係emp を属性efname,enameで昇順で分類し、結果を結果関係名testに格納する為のコマンドは次の様である。

```
srt emp on efname, ename in a to test;
```

⑦prt コマンド

関係の組を画面に表示するコマンドである。

prt　< 関係名 > [(<属性名>[/[< タイプ>] <長さ>]

{, <属性名>[/[< タイプ>] <長さ>] })];

[例]
関係emp の従業員番号(eno),氏名(jname) を表示するコマンドは次の様である。

```
prt emp ( eno, jname );
```

氏名を10桁表示するのは次の様である。

```
prt emp ( eno, jname/10 );
```

注意) 出力行は80桁までしか出力できない。これ以上を出力しようとするとエラーメッセージが表示される。

⑧　lst コマンド

関係の組を画面に表示するコマンドである。

lst　< 関係名 > [(<属性名>[/[< タイプ >] <長さ>]

{,　<属性名>[/[< タイプ >] <長さ>] })];

[例]

関係emp の従業員番号(eno)、氏名(jname) を出力するコマンドは次の様である。

lst emp (eno, jname);

氏名を10桁出力するのは次の様である。

lst emp (eno, jname/10);

注意) 出力行は８０桁までしか出力できない。これ以上を出力しようとするとエラーメッセージが表示される。

⑨　dsp コマンド

関係及び属性定義の情報を画面に表示する為のコマンドである。

dsp〈テーブル名〉〔 ,〈テーブル名〉〕;

〔例〕

関係定義情報を表示するコマンドは次の様である。

dsp reltbl;

関係定義テーブル

関係番号	関係名	次数	生成年月日	組数	組長	クラスタ	索引数	格納モード	先頭組識別子	最終組識別子
rel	rel_name	deg	c_date	card	width	c	n	s	h_ptr	t_ptr

属性定義情報を表示するコマンドは次の様である。

dsp attbl;

属性定義テーブル

関係番号	属性番号	属性名	タイプ	NULL	桁長	索引	重複	クラスタ	ハッシュ	根ページ	葉の先頭ページ	葉の最終ページ
rel	an	att_name	tp	na	le	in	un	cl	ha	indpt	fiptr	liptr

⑩ dmpコマンド

結果関係，結果属性，組変数の情報を画面に表示する為のコマンドである。

dmp〈テーブル名〉〔，〈テーブル名〉〕;

〔例〕

組変数定義情報を表示するコマンドは次の様である。

dmp rtbl;

組変数定義テーブル

組変数名	関係番号	関係名	目標リスト	主制限リスト	非主制限リスト
var_name	rel_id	rel_name	tlist	prstr	nptr

結果関係定義情報を表示するコマンドは次の様である。

dmp rrtbl;

結果関係定義テーブル

結果関係番号	結果関係名	次数	組長	組数	0_treeへのポインタ	タイプ	先頭の組識別子	最終の組識別子
relN	rel_name	deg	width	card	qptro	qptrm	ty	hptr	tptr

結果属性定義情報を表示するコマンドは次の様である。

dmp ratbl;

結果属性定義テーブル

関係番号	属性番号	属性名	タイプ	桁長	オフセット
rel_id	at_no	att_name	tp	ln	off

⑪ exe コマンド

コマンドを実行させるコマンドである。

exe〔行番号〕;

行番号が指定された場合，指定行にあるコマンドを実行する。指定が無い場合は直前のコマンドを実行する。

exeを使って実行させるコマンドとしては，get,app,rep,del がある。lst,prt,srt,crt は exe によって再実行される。

〔例〕

```
#00020 get [e.all];
#00030 exe;                          get [e.all]を実行する。
#00040 get [e.eno; e.efname] e.eno > 100;
#00050 get [e.eno; e.jname]  e.eno < 100;
#00060 exe 40;                       get [e.eno; e.efname] e.eno > 100を実行する。
#00070 exe 50;                       get [e.eno; e.jname]  e.eno < 100を実行する。
```

⑫ els コマンド

コマンドバッファ（ＱＢＦ）内の内容を表示するコマンドである。

els〔〈行番号〉,〔〈行番号'〉〕〕;

指定範囲のコマンドを画面に表示する。

〔例〕

上の場合で els コマンドを実行すると次の様に画面に表示される。

```
#00020 get [e.all];
#00040 get [e.eno; e.efname] e.eno > 100;
#00050 get [e.eno; e.jname]  e.eno < 100;
```

コマンドバッファ（ＱＢＦ）に保存されるコマンドは次のコマンドである。

crt,get,app,rep,del,srt,prt,lst

他のコマンドは実行後はコマンドバッファから消去される。

i） 1行表示の場合

```
els 2 0 ;
#00020 get [e.all];
```

ii） 範囲指定の場合

```
  els 20. 40;
#00020 get [e.all];
#00040 get [e.eno; e.efname] e.eno > 100;
```

⑬ ern コマンド

コマンドバッファ（ＱＢＦ）の行番号をつけなおす為のコマンドである。

ern;

１０から１０単位に行番号をつけなおす。

［例］

```
行番号を付け直す前のコマンドバッファの内容
#00020 get t1.....
#00040 srt  ......
#00050 prt  ......
#00070 get t2.....
#00120 get t3.....
行番号を付け直した後のコマンドバッファの内容
#00010 get t1.....
#00020 srt  ......
#00030 prt  ......
#00040 get t2.....
#00050 get t3.....
```

⑭ edl コマンド

コマンドバッファ（ＱＢＦ）の指定行を消去する為のコマンドである。

edl〈行番号〉;

［例］

```
消去前のコマンドバッファの内容
#00020 get t1 ...
#00040 srt  .....
#00050 prt ......
#00070 get t2....
#00090 get t3 ...
行番号70を消去した後のコマンドバッファの内容
#00020 get t1 ...
#00040 srt  .....
#00050 prt ......
#00090 get t3 ...
```

⑮ erp コマンド

コマンドバッファ（ＱＢＦ）の内容を修正する為のコマンドである。

erp〈行番号〉;

指定行を修正する為に修正前の内容を表示し，修正内容を入力させる。

［例］

```
修正前のコマンドバッファの内容
#00020 get t .....
```

結果関係名を変更したい場合、次の様にerp コマンドを使用する。

```
#00040 erp 20;
    20 get t ......          変更前の行内容
 00020 get test ...          変更する行内容を入力する
```

⑯ ein コマンド

コマンドバッファ（ＱＢＦ）にコマンド行を追加する為のコマンドである。

ein〈行番号〉；

[例]

```
追加前のコマンドバッファの内容
#00030 get [e.eno; e.jname ]
#00040 e.eno > 100 and
#00050 e.esex = "m";
行番号40と50の間にコマンド行を挿入するには次の様にコマンドを入力する。
#00430 ein 45;
 00045 prt ......
追加された後のコマンドバッファの内容
#00030 get [e.eno; e.jname ]
#00040 e.eno > 100 and
#00045 e.efname = "yokotsuka" and
#00050 e.esex = "m";
```

⑰ sav コマンド

作成したコマンドをデバイス上のファイルに書き込む為に使用されるコマンドである。

sav〈ファイル名〉〔行番号〔，行番号〕〕；

コマンドバッファ（ＱＢＦ）内の行番号で save する範囲を指定する。指定された範囲のコマンドと検索・更新で参照している組変数を宣言する var コマンド文がファイルに書き込まれる。

[例]

コンドバッファの内容を外部記憶にセーブする為のコマンドは次の様である（外部ファイル名；ＱＵＥＲＹ１）。

sav ＱＵＥＲＹ１；

この場合コマンドバッファの全部の内容をファイルQUERY1 に書き込む。この他にコマンドバッファの行番号を指定してファイルに書き込むことができる。

sav QUERY2　20，100；

コマンドバッファの行番号が２０から１００までのコマンドをファイルQUERY2に書き込む。

注意）savでファイルが生成出来ない場合，エラーメッセージ”ファイルが書き込み出来ない。”と表示される。このメッセージが表示されたら，別のファイル名でもう一度savを実行してみる。それでも書き込みが出来なければ，容量がないかどうか調べてみる。

⑱　lodコマンド

コマンドをsave　したファイルを続み込み，コマンドバッファ（QBF）内にload　する為のコマンドである。

lod〈ファイル名〉；

〔例〕

savコマンドでセーブしたコマンドをコマンドバッファに読み込む為のコマンドは次の様である。

lod QUERY1；

注意）lodでファイルが読み込み出来ない場合，エラーメッセージ”ファイルが読み込み出来ない。”と表示される。このメッセージが表示されたら，savしたファイル名を確認してみる。

⑲　fdlコマンド

コマンドをsave　したファイルを消去する為に使用するコマンドである。

fdl〈ファイル名〉；

〔例〕

セーブしたコマンドファイルを消去する為のコマンドは次の様である。

fdl QUERY1；

⑳　endコマンド

Granadaを終了させる為のコマンドである。

end〔；〕

C．フォームインタフェース

① フォームインタフェースの使用法

フォームインタフェースはGranada本体とは独立である。フォームインタフェースを使用するコマンドとしてはframeがある。frameをコールするとフレールの更新，検索・更新，自動検索・更新が選択出来る画面が表示される（図4.20）。初期の時点では，フレームの更新が反転表示されている。利用者は処理したい機能の表示をカーソルを移動させて（→と←を使い）選択する。自動検索・更新ではフレームをシステムが自動的に作成して検索・更新を提供する。

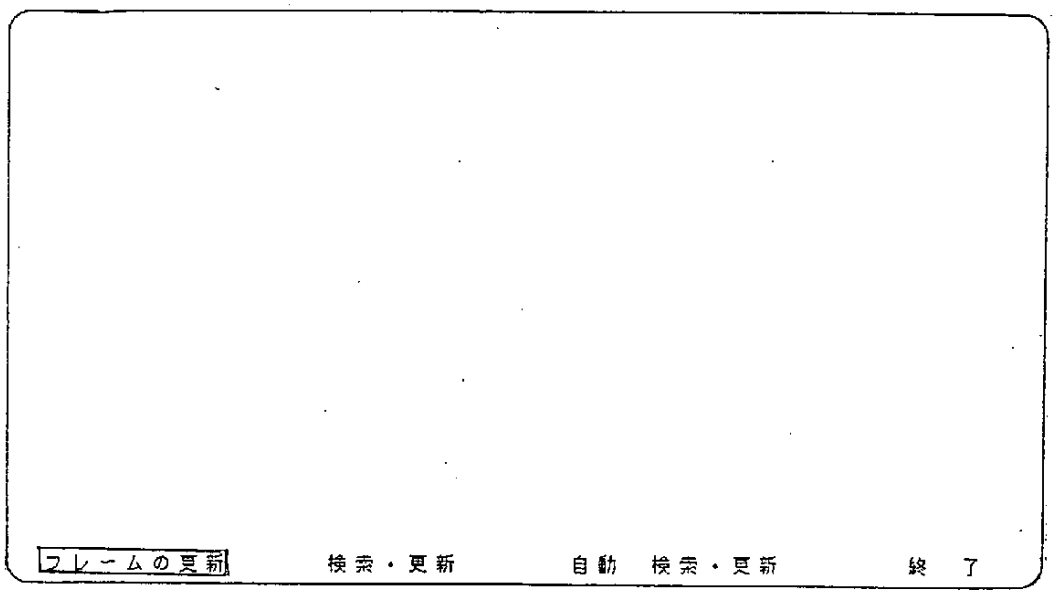


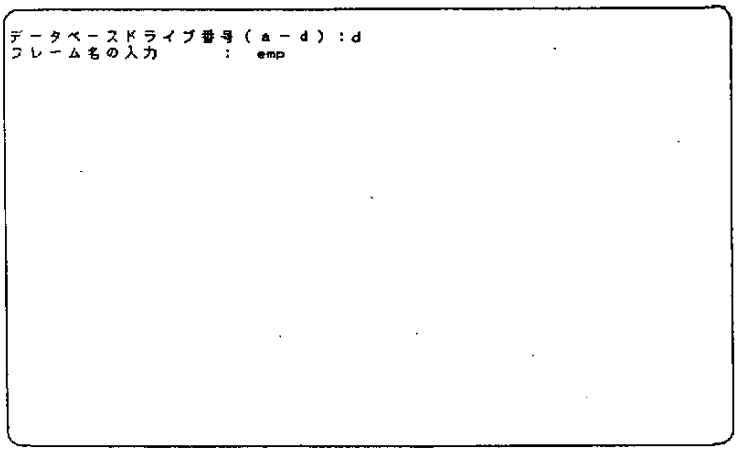
図4.20 フォーム処理の初期画面

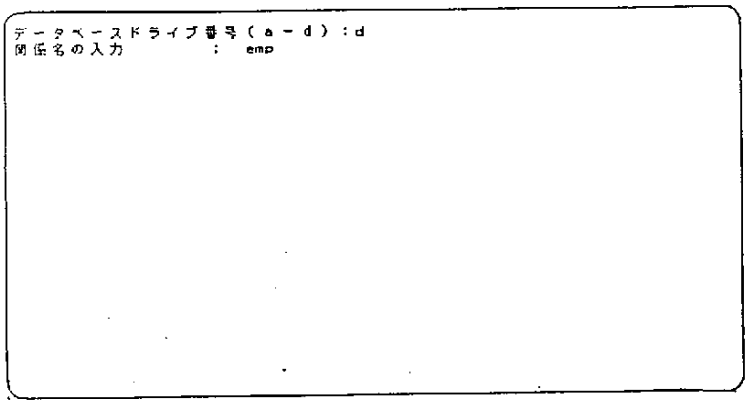
フレームの更新については，フォーム処理モヂュールを参照されたい。

② 検索・更新

② 検索・更新

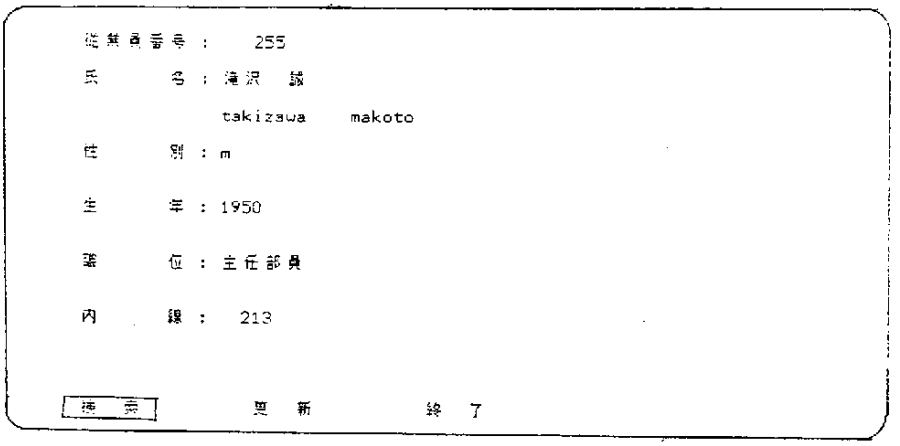
検索・更新を選択するとまづ最初に図の様な画面が表示される。データベースが格納されているドライブ番号（Ａ－Ｄ）を入力する。次に検索・更新するフレーム名を入力する（自動の場合は関係名の入力となる)。データベースとフレームの定義情報を読み込み，指定フレームに対応する関係の最初の組を画面に表示する。

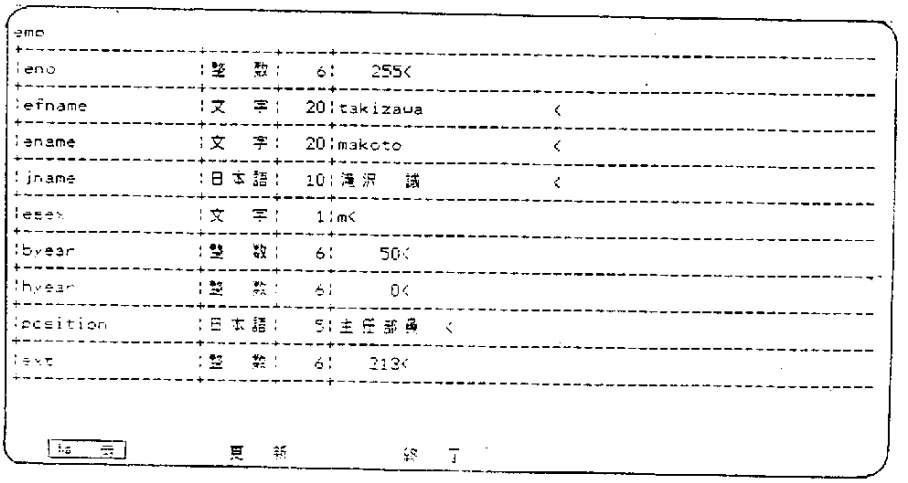




（自動の場合）

選択したフレームに対応する関係の先頭の組の内容を表示する。次に検索を行うか，更新を行うかを選択する。選択はカーソルを移動させ，機能の表示を反転させ復改キーを押すことで行われる。

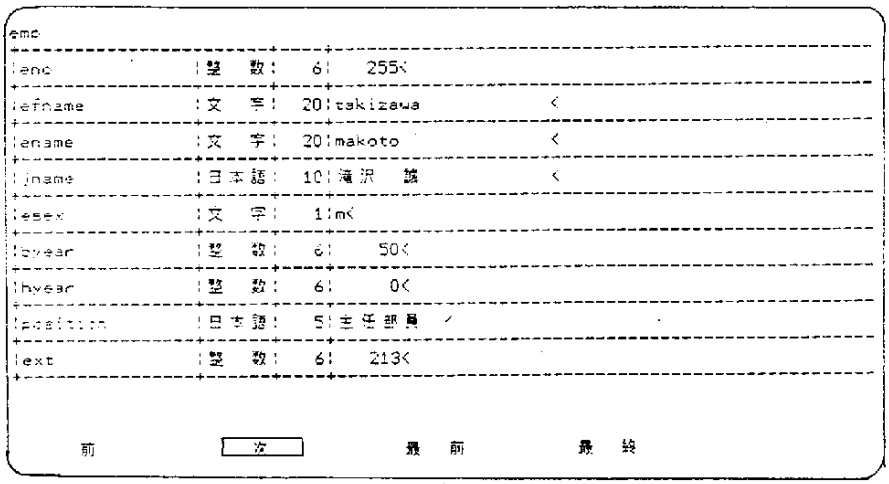




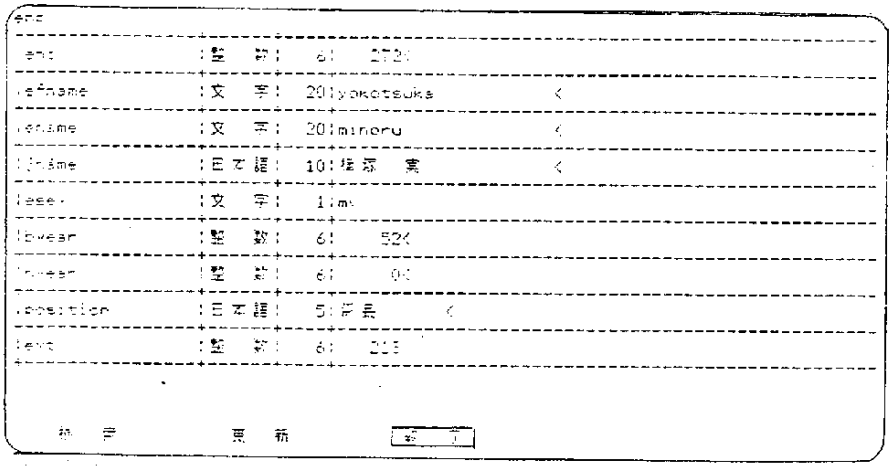
（自動の場合）

(i) 検索

検索を選択すると図の画面になる。現在表示されている組の１つ前の組，１つ後の組，関係の先頭の組又は最終の組のいづれかを選択する。初期時は，１つ前の組を検索する機能が反転表示されている。



機能が選択されると該当する組が表示される。



検索を終了したい場合は終了を選択する。

(ii) 更新

更新を選択すると，画面上のデータの入力モードになる。追加または修正の場合，データを順次入力する。入力が終了するとデータ入力が紙了したか確認を行う。次の様な確認画面が表示される。

```
emp
+----------+------+---+-----------------+
|eno       |整 数 | 6|272  <            |
+----------+------+---+-----------------+
|efname    |文 字 |20|yokotsuka        <|
+----------+------+---+-----------------+
|ename     |文 字 |20|minoru           <|
+----------+------+---+-----------------+
|jname     |日本語| 10|横塚 実          <|
+----------+------+---+-----------------+
|esex      |文 字 | 1|m<                |
+----------+------+---+-----------------+
|byear     |整 数 | 6|52   <           |
+----------+------+---+-----------------+
|hyear     |整 数 | 6|    0<           |
+----------+------+---+-----------------+
|position  |日本語| 5|係長     <       |
+----------+------+---+-----------------+
|ext       |整 数 | 6|213  <           |
+----------+------+---+-----------------+

  DATA  OK (y/n) ?
```

次に更新機能（追加・修正・削除）の選択画面が表示されるので，実行する機能を選択する。

```
emp
+----------+------+---+-----------------+
|eno       |整 数 | 6|  272<           |
+----------+------+---+-----------------+
|efname    |文 字 |20|yokotsuka        <|
+----------+------+---+-----------------+
|ename     |文 字 |20|minoru           <|
+----------+------+---+-----------------+
|jname     |日本語| 10|横塚 実          <|
+----------+------+---+-----------------+
|esex      |文 字 | 1|m<                |
+----------+------+---+-----------------+
|byear     |整 数 | 6|   52<           |
+----------+------+---+-----------------+
|hyear     |整 数 | 6|    0<           |
+----------+------+---+-----------------+
|position  |日本語| 5|係長     <       |
+----------+------+---+-----------------+
|ext       |整 数 | 6|  213<           |
+----------+------+---+-----------------+

  追 加        変 更        削 除        再入力        終 了
```

（自動の場合）

4.3 評　価

Granada Ver1.0 の性能を他の超小型計算機用のＤＢＭＳと比較するとともに，ＰＣ９８０１とＳＵＮ２／１２０での性能を比較する。他のＤＢＭＳとしてはCONDOR　Ｓ２０をＰＣ９８０１上で実働させて比較する。

4.3.1　他ＤＢＭＳの比較

A．評価環境

ＰＣ９８０１用のDBMS　CONDOR　Ｓ２０とGranadaとを比較する。評価では同一データベースを使用して，GranadaとCONDOR　Ｓ２０でのアクセスの性能及びデータの格納効率を比較する。データベースとしてはＴＥＳＴ(ＥＮＯ，ＥＦＮＡＭＥ)，ＥＮＯ／２バイトの整数，ＥＦＮＡＭＥ／２０バイト文字列で，索引がＥＮＯに対して付けられている。データはハードディスクに格納してある。Granadaでは実行時間の測定の為，システム内の時刻によってその実行時間を測定する。CONDORに対しては，コマンドの入力から実行が終了するまでの時間をストップウォッチを使い測定する。

B．評価方針

データベースのデータ件数を１００，２００，４００，６００，８００，１０００と変えて，データの格納効率，順次アクセスの処理効率，索引の格納，索引のアクセスの処理効率，分類の処理効率等を測定する。

C．評価結果

Granadaは可変長レコード格納によって，表4.7ａ）でも明らかな様にCONDOR　Ｓ２０に比べて約６割の容量でデータを格納出来る。しかし順次アクセス〔表4.7ｂ）〕を見ると，圧縮を復元する為のＣＰＵ負荷の為にアクセスの性能が劣っている。関係に索引を付けるコストは表4.9の様である。又索引を格納するのに必要な容量は表4.7ｇ）の様である。

関係に索引をつけた場合，索引を使用したアクセスではデータ件数によらず一定の値でアクセスしている〔表4.7ｄ）〕。索引を持つ関係に対する制限条件つきの結合処理の結果も同様に一定の値でアクセスしている〔表4.7ｅ）〕。

表 4.7　評価（Granada と CONDOR）

a）ファイル容量（データ）　〔単位：バイト〕

組数（組長22バイト）	100	200	400	600	800	1000
Granada	2048	3072	6144	9216	11264	14336
CONDOR	2816	5376	10624	15744	20992	26246

b）順次アクセス

Granada get〔t.all〕;

CONDOR project tes eno efname

組数（組長22バイト）	100	200	400	600	800	1000
Granada	30秒	59秒	120秒	180秒	260秒	299秒
CONDOR	16秒	17秒	19秒	20秒	21秒	22秒

c）索引付けコスト

Granada ind test on eno ;

CONdor index test using eno

組数（組長22バイト）	100	200	400	600	800	1000
Granada	15秒	41秒	93秒	146秒	200秒	240秒
CONDOR	13秒	14秒	17秒	21秒	27秒	32秒

d）索引を使用したアクセス

Granada get〔t.all〕t.eno＝272;

CONDOR select test where eno＝272

組数（組長22バイト）	100	200	400	600	800	10000
Granada	1秒	1秒	1秒	1秒	1秒	1秒
CONDOR	19秒	19秒	19秒	19秒	19秒	19秒

e）結合処理コスト

```
Granada var(t,test)(t1,test);
        get(t,eno;t1,efname)t,eno = t1,eno and t,eno = 272;
CONDOR  select test where eno = 272
          join test result using eno
```

組数（組長22バイト）	100	200	400	600	800	1000
Granada	2秒	2秒	2秒	2秒	2秒	3秒
CONDOR	42秒	43秒	44秒	44秒	44秒	44秒

f）ソート処理コスト

```
Granada srt test on eno,efname to A;
CONDOR  srt test using eno efname
```

組数（組長22バイト）	100	200	400	600	800	1000
Granada	6秒	10秒	20秒	30秒	42秒	64秒
CONDOR	18秒	20秒	24秒	29秒	40秒	49秒

g）ファイル容量（索引）　　〔単位：バイト〕

組数（組長22バイト）	100	200	400	600	800	1000
Granada	1024	3072	6144	9216	13312	14336
CONDOR	2432	3068	7808	11008	14592	18176

4.3.2　PC9801とSUN2／120との比較

A．評価目的

Granada DBS の性能とボルトネックを明らかにする。

B．評価方法

PC9801とSUN2／120 に対して，同一データベースを用いて行う。データベースとしては tnnn(no,name)〔nnnはデータ件数を表す〕no は2バイト整数，name は 100 バイト文字列である。データはランダム関数を用いてプログラムで自動生成されたものを使用する。

Granada DBMS 内に時間取得関数（SUN2／120ではgetitime 関数，

PC9801ではシステムコールによる時間取得関数）を埋め込み，モヂュールの実行時間を測定する。又測定の際は，各コマンド毎に，Granadaを立ち上げて実行する（メモリバッファをクリアする為）。

評価は次の項目について行う。

i）逐次アクセスの性能

関係の組を先頭から逐次アクセスする場合の実行時間を測定する。又，文字列(name) 比較条件を付けた場合の逐次アクセスの実行時間も測定する。

ii）ソート処理の性能

関係の組をソートする時の実行時間を測定する。

iii）索引付きの関係間の結合処理の性能

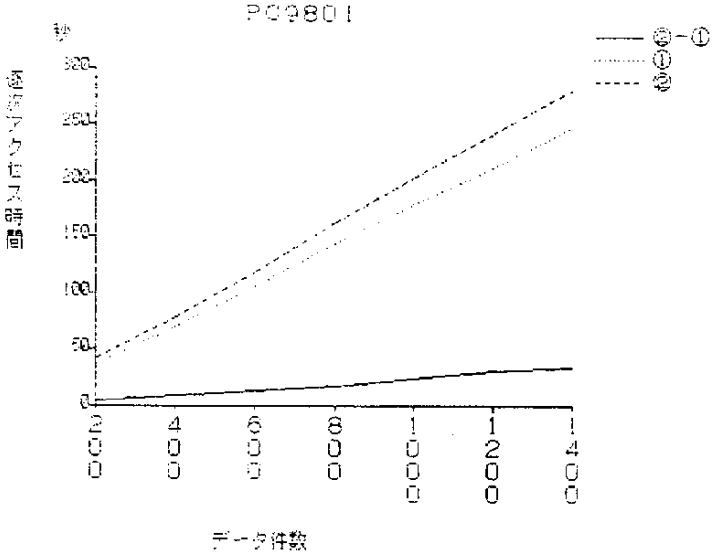
関係の属性 no に索引がある場合の no に対して結合処理時間を測定する。

C．評価結果

i）逐次アクセスの性能

逐次アクセスの実行時間をデータ件数毎にプロットした図が図4.21である。①は条件の無い逐次アクセスの結果で，②は name に10バイトの条件を付けた場合の逐次アクセスの結果である。②－①は条件の判定等にかかった時間である。

PC9801では条件の判定に1組当たり約0.03秒要している。また1組の検索では約0.17秒要している。SUN2／120では条件の判定に1組当たり約5ミリ秒要している。また1組の検索では約50ミリ秒要している。


PC9801
秒
逐次アクセス時間
300
250
200
150
100
50
0
2000
4000
6000
8000
10000
12000
14000
データ件数
②−①
①
②

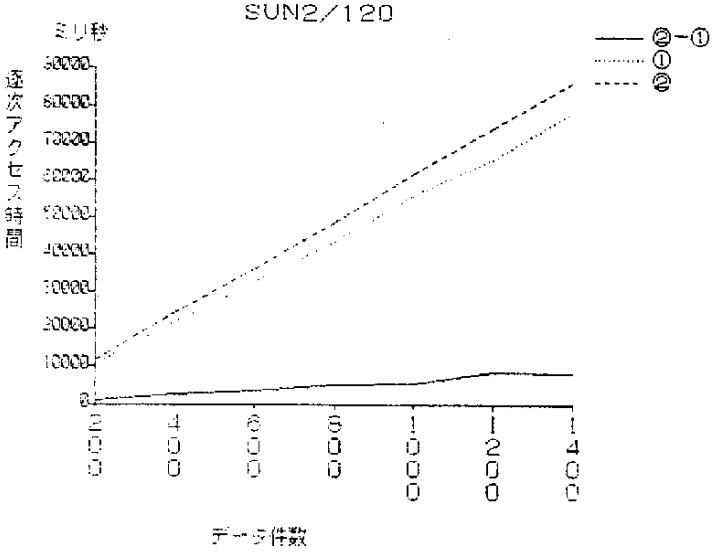

SUN2／120
ミリ秒
逐次アクセス時間
80000
70000
60000
50000
40000
30000
20000
10000
0
2000
4000
6000
8000
10000
12000
14000
データ件数
②−①
①
②

図 4.2.1　逐次アクセス

ii） ソート処理の性能

ソート処理はソートするキーをメモリバッファに入り切れるだけ入れ，その中でソートし連を生成している。また生成された複数の連を併合してソードを行う。

PC9801でのソート処理の実行時間と，SUN2／120でのソート処理の実行時間は，図4.22の様である。PC9801の結果は，データ件数に対して二次曲線として近似している。SUN2／120の結果は，データ件数に対して線形に増大している。

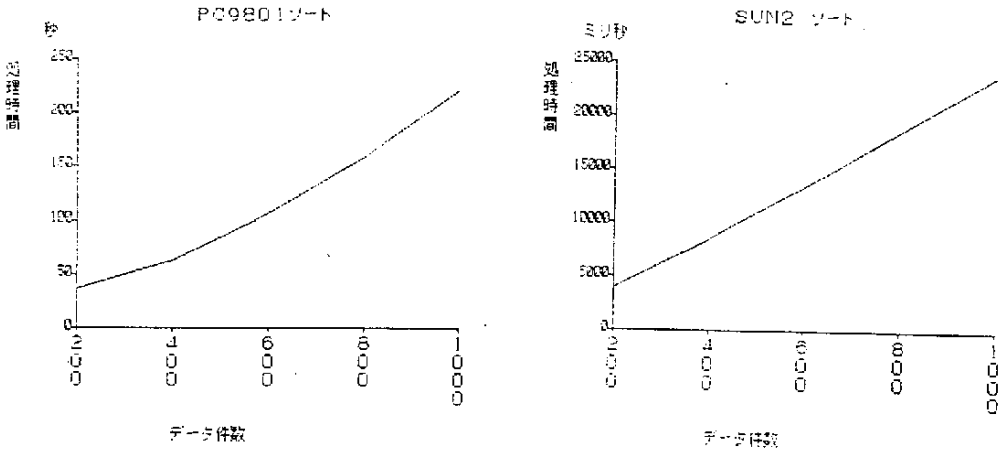


図4.22　ソート処理

iii） 索引付きの関係間の結合処理の性能

索引を付けた関係間の結合処理の実行時間は図4.23の様である。索引がついている為実行時間は分のオーダーで終了する。同じ問合せで索引が無い場合は，実行時間は時間のオーダーになる。

二変数（一属性結合）

```
var (t, tnnn) (t1, tnnn);
get [t.no; t1.name] t.no = t1.no;
```

ＰＣ９８０１の結果　　［単位：秒］

	検索件数	t200 - t200	t200 - t400	t200 - t600	t400 - t400
5HD	検索時間	78	86	89	164
8FD	検索時間	91	116	128	209
5FD	検索時間	95	109	124	194

ＳＵＮ２／１２０の結果　　［単位：ミリ秒］

検索件数	t200 - t200	t200 - t400	t200 - t600	t400 - t400
検索時間	31080	31760	26820	60500

図 4.2 3　結　合　結　果

4.3.3　今後の課題－分散型Granada

Granadaの可変長による格納で格納効率は他のパーソナルＤＢＳよりも良い事が明らかになった。しかし関係の組を先頭から逐次アクセスするのに時間を余計に要している。これは，圧縮・復元等にＣＰＵ負荷が係るためと考えられる。超小型計算機では物理Ｉ／ＯよりもＣＰＵ負荷の方が実行時間に影響を与えていると考えられる。この為ＣＰＵ負荷を軽減することを検討する必要がある。その一つとして関係の組の格納を固定長で格納することを検討する。また検索条件の判定もＣＰＵ負荷を大きくしている。

ソート処理などでは，物理Ｉ／Ｏの回数が増大しておりメモリバッファ管理の方法を改良することを検討する。

結合処理では，索引がある場合では実行が分のオーダーなのに対して，索引がない場合は実行は時間のオーダーになっている。この実行時間を短縮する為，索引のない場合は一時的に結合処理の間だけ索引を付けて処理を実行

させる方法と，関係をソートしてから併合する方法とを検討している。

4.4 ま と め

超小型計算機と大型機をＬＡＮで結合するシステムＬＩＳの主要ＤＢＭＳとして超小型機上に従来の大型機上のＤＢＭＳと同等の機能を持つパーソナルＤＢＭＳ Granadaを開発した。GranadaはＰＣ９８０１上に実現され，次にＳＵＮ２／１２０に移植された。現在はＰＣ９８０１とSUN2／１２０上で実働している。

Granadaの性能を向上させる為，現在次のことを検討している。

(1) データの圧縮と復元の為のＣＰＵ負荷を軽減する。

(2) 物理Ｉ／Ｏ回数を減ら様なメモリバッファ管理。

(3) 索引の無い結合処理では一時的に索引をつけて結合処理を行う。

またGranadaの機能強化としては，

(1) 検索条件の機能強化

- 文字列の部分一致
- 自由な形式の検索条件

(2) コマンドの編集機能

等を行う。

今後の開発としてマルチユーザー化及びリカバリ処理の追加を検討している。又，ＬＡＮで結合されたパソコン上の分散型Granadaの開発を進めている。分散型Granadaでは，各サイトのDB（subDB）から他サイトのＤＢに必要なデータを抽出して使用する形態を考えている〔図4.２４〕。

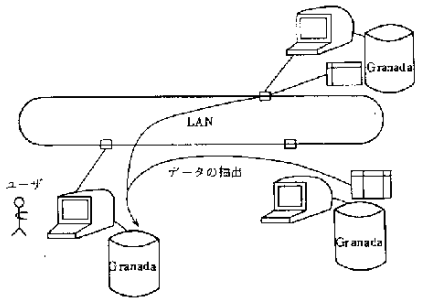


図4.２４　分散型Granada

その他に，他機種への移植としてF9450ＩＩのＣＰ／M86への移植を行う予定である。

5. まとめと今後の課題

本年度は，本事業の第２年度として，以下の作業を行った。

(1) パソコン用関係型ＤＢＭＳ Granadaの開発，改良，評価

(2) ローカル情報システム（ＬＩＳ）の基本通信手段としてのトランスポート層のプロトコルＴＣＣＰの実現

(3) ＴＣＣＰの上位のＤＤＢＳプロトコル（ＤＤＢＰ）の基本設計

(4) 異種なＤＢＳ間でのインタオペラブルな利用を行させるための統合化技術の検討

(1)のパソコン用の関係型ＤＢＭＳとしてのGranadaは現在ＰＣ－９８００（ＭＳ－ＤＯＳ）に加えて，Sun２／１２０（Unix 4.2 bsd）で実働している。機能的には，ミニコン用の関係型ＤＢＭＳ Ingressの利用者言語Quel と同等な非手続的な利用者言語ＲＱＬが利用者に提供される本格的な関係型ＤＢＭＳである。関係の結合，射影，制限といった関係演算に加えて，group－by付きの集約演算（例．sum，count，aver）を行うことが出来る。更に，フォーム形式の利用者インタフェースを備えている。Granadaは，ＬＩＳの各利用者の作業用のＤＢＭＳとなるものである。現在，Granadaは，データの順次読出しがパフォーマンスのネックとなっている。これは，データを圧縮して格納している為に，この圧縮復元に，ＣＰＵを使用されてしまうからである。現在，この改良を試みている。

ＬＩＳで，複数のＤＢＳ間での通信を行うときに，ＤＢＳが一体となった通信が求められる。Ethernet，トークンリングの様なＬＡＮでは，１つの実体から発信された情報が全実体に届けられるという放送通信が，データリンク（正しくは，ＭＡＣ（媒体アクセス制御））レベルで備わっている。しかし，これは，信頼性が保障されず，任意の実体のグループに対して放送出来ない。この為に，任意の実体の集合に対して，この中の実体間で，高信頼な放送通信を，順序を保存し，かつ非同期に行えるトランスポート層のプロトコル，ＴＣＣＰ（トランスポート群制御プロトコル）を，Ethernetのデータリンクサービス上に実現した。ＴＣＣＰは，Ethernetに結合されたＵ－１２００とSun２／１２０に，Unixの利用者プログラムとして，Ｃ言語で実現されている。現在はＬＡＮと計算機を，９６００bpsのシリアルポートで結合しているために，ここが通信のボトルネックとなってしまっている。この為，Ｕ－１２００は，ＩＥＥＥ－４８８で，Sun２／１２０は，SunのEthernetコントローラを用いて直結する方法を検討している。

(3)と(4)については，基本的な問題　の整理を行った。(3)については，この結果をもとにして，来年度実現予定である。

参 考 文 献

〔Bernstein 81a〕 Bernstein, P. A. and N. Goodman, "Concurrency Control in Distributed Database Systems," Computing Surveys 13, 2, June, 1981, pp. 185−221.

〔Bernstein 81b〕 Bernstein, P. A., et al., "Using Semi−Joins to Solve Relational Queries," JACM, Vol. 28, No. 1. 1981, pp. 25−40.

〔Chan 83〕 Chan, A., et al., "Overview of an Ada Compatible Distributed Database Manager," Proc. of the ACM SIGMOD' 83, 1983, pp, 228−237.

〔Chen 76〕 Chen, P. P, −S., "The Entity−Relationship Model−Toward a Unified View of Data," ACM TODS, Vol, 1, No. 1, 1976, pp. 9−36.

〔CODASYL 73〕 Codasyl DDL Committee, "CODASYL Data Description Language," Journal of Development, 1973.

〔CODASYL 78〕 Codasyl DDL Committee, "Report of the CODASYL Data Definition Language Committee," Information Systems, Vol. 3, No. 4, 1978, pp. 247−320.

〔Codd 70〕 Codd, E. F., "A Relational Model of Data for Large Shared Data Bank," CACM, Vol. 13, No. 6, 1970, pp. 337−387

〔Codd 82〕 Codd, E. F., "Relational Databases: A Practical Foundation for Productivity," CACM, Vol. 25, 1982, pp. 109−117.

〔DARPA 81〕 DARPA Information Processing Techniques Office, "INTERNET PROTOCOL",

Information Sciences Institute University of Southern California, September, 1981

[Date 81] Date, C. J., "Introduction to Database Systems," (2nd. ed.), Addison-Wesley, 1981,

[Dayal 82] Dayal, U., et al., "Query Optimization for CODASYL Database Systems," Proc. of the ACM SIGMOD, 1982, pp. 138-150.

[Denning 76] Denning, D. P, "A LatticeModel of Secure Information flow", CACM, May, 1976

[Denning 82] Dcnning, D. P, "Cyptography and Data Security", Addison-Wesley, 1982

[Emden 81] Emden, M H, etal, "Equations Compared with Clauses for Specification of Abstract Data Types,"ADVANCES IN DATABASE THEORY, Vol. 1, Plenum Press, 1981

[Eswaran 76] Eswaran, K. P., J. N. Cray, R. A. Lorie and I. L. Traiger. "The Notions of Consistency and Predicate Locks in a Database System, "Communications of the ACM 19, 11 (Nov. 1976)

[Goguen 79] Goguen. J. A. et al. "An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types," Current Trends in Programming Methodogy IV: Data Structuring (R. Yeh ed.), Prentice Hall, 1978, pp. 80-144

[Goodman 79] Goodman, N., et al., "Query Processing in SDD-1 : A System for Distributed Databases," CCA-79-06, CCA, 1979, pp. 1-57.

[Gouda 81] Gouda, M., et al.,"Optimal Semi-Join Schedule for Query Processing in Local Distributed Systems," Proc. of the ACM SIGMOD. 1981, pp. 164-175.

[Gray 81] Gray, J., "Notes on Database Operating Systems," Operating Systems: Advanced, Cource, Springer-Verlac, 1981.

[Held 76] Held, C., et al., "INGRES-A Relational Database Systems," AFIPS Conf, Proc., 1976, pp. 409-416.

[Hevner 78] Hevner, A., et al., "Query Processing on a Distributed Databases," Proc. of the 3rd Berkeley Workshop, 1978, pp. 91-107.

[Hopcroft 69] Hopcroft, J, E., and Ullman, J, P., "Formal Languages and Their Relation to Automation," Addison-Wesley, 1969.

[Lamport 78] Lamport, L., "Time, Clocks and the Ordering of Events in a Distributed System," Communications of the ACM, 21, 7, 1978

[Lcbihan 80] Lc Bihan, J., et al., "SIRIUS: A French Nationwide Project on Distributed Data Base," Proc. of the 6th VLDB, 1980.

[Metcalfe 76] Metcalfe, R. M., et al., "ETHERNET Distributed Packet Switching for Local Computer Networks," CACM, Vol. 9, 1976, pp. 395-404.

[Noguchi 83] Noguchi, S., et al., "情報ネットワークの理論," 岩波講座 情報科学 Vol. 5, 1983.

[Onuege 83] Onuege, E., et al., "Local Query Translation and Optimization in a Distributed System," AFIPS Conf. Proc., 1983, pp. 229-239.

[Rothnie 80] Rothnie, J. B., et al., "Introduction to a System for Distributed Databases(SDD-1)," ACM TODS, Vol. 5, 1980, pp. 1-17

[Smith 81] Smith, J. M., et al., "Multibase-Integrating Heterogeneous Distributed Databases Systems," AFIPS Conf. Proc., 1981, pp. 487-499.

〔Suzuki 79〕 Suzuki, K., et al., "DEIMS-2について," IPSJ DBMS 研 11-3, 1979.

〔Takizawa 78〕 Takizawa, M., et al., "Resource Integration and Sharing on Heterogeneous Resource Sharing Systems," Proc. of the ICCC' 78, 1978, pp. 253-258.

〔Takizawa 79〕 Takizawa, M., and Hamanaka, E., "The Four-schema Consept as Gross Architecture of Distributed Databases and Heterogeneous Problems," JIP(IPSJ), Vol. 2, No. 3, 1979, pp. 134-142

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

〔Takizawa 81〕 Takizawa, M., "分散型データベースシステムJDDBS-IIと通信処理," 電子通信学会AL81-22, 1981.

〔Takizawa 82a〕 Takizawa, M., "Distribution Problems in Distributed Databases," JIP(IPSJ), Vol. 5, No. 3, 1982, pp. 139-147.

〔Takizawa 82b〕 Takizawa, M., Yokotsuka, M., et al., "CODASYLデータベースシステムに対する関係インタフェースシステム(LDP-V 1.5)の設計と実現," 情報処理学会 論文誌, Vol. 23, No. 3, 1982, pp. 665-675

〔Takizawa 82c〕 Takizawa, M., "放送通信機能を用いた分散型データベースシステムの通信処理の実現について," 情報処理学会 DBMS研31, 1982.

〔Takizawa 82d〕 Takizawa, M. and Yokotsuka, M., "ローカルデータベースプロセッサ(LDP)の評価," データセマンティクスの理論と実際に関する研究, 京大数理解析研講究録461, 1982, pp. 127-150.

〔Takizawa 83a〕 Takizawa, M. and Noguchi, S., "CODASYLデータベースシステムに対する非手続的更新インタフェース

の基本概念," 情報処理学会 論文誌, Vol. 24 No. 6, 1983, pp.

〔Takizawa 83b〕 Takizawa, M., "Distributed Database System－JDDBS," JARECT Vol. 7, Computer Science and Technologies (Kitagawa, T., ed.), Ohmsha and North－Holland, 1983, pp. 262－283.

〔Takizawa 83c〕 Takizawa, M., Yokotsuka, M., and Noguchi, S., "分散型データベースシステムに於ける放送通信を用いた通信処理方式," 情報処理学会「ローカルエリアネットワーク」シンポジウム報告書, 1983, pp. 211－218.

〔Takizawa 84a〕 Takizawa, M. and Noguchi, S., "CODASYL DML上の非手続グラフ問合せの設計と実現," 情報処理学会 論文誌, Vol. 25, No. 5, 1984, pp. 764－774.

〔Takizawa 84b〕 滝沢, 横塚, 盛屋, 永田, 竹内, 野口, "LANを用いた分散型データベースシステムLISの通信処理手順について," 情報処理学会「LAN／マルチメディアの応用と分散処理」シンポジウム報告書, Oct. 1984, pp. 77－84.

〔Takizawa 84c〕 滝沢, "LANを用いた分散型データベースシステム," 日経コンピュータ, 10月29日号, 1984, pp. 115－125.

〔Toan 81〕 Toan, N, G., "Distributed Query Management for a Local Network Database System," Proc. of the 2nd Distributed Computer Systems, 1981, pp. 188－196.

〔Vassiliou 80〕 Vassiliou, Y., et al, "DBMS Transation Translation," Proc. of the COMPSAC80, 1980, pp. 89－96.

〔Wilmes 83〕 Wilmes, P, F., et al., "I with I were over there: Distributed Execution Protocols for Data Definition in R*," Proc. of the ACM SIGMOD,

1983. pp. 238-242.

[Wong 77] Wong, E., "Retrieving Dispersed Data from SDD-1: A System for Dispersed Databases," Proc. of the 2nd Berkeley Workshop, 1977, pp. 217-235.

[Yamazaki 82] Yamazaki, H, et al., "A Proposal for Broadcast Arehitecture Network(BANET)," Proc. of the ICCC'82, 1982.

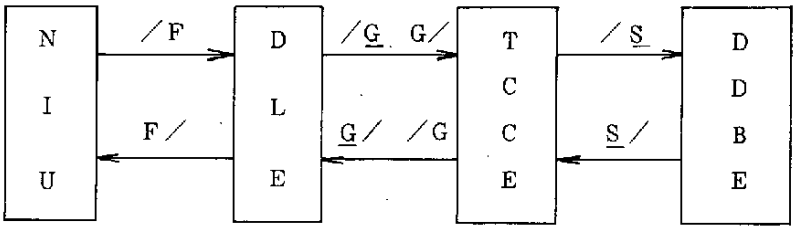
[Yao 76] Yao, S. B., "Attribute-based Model,," ACM TODS, 1979.

[Zaniolo 79] Zaniolo, C., "Design of Relational View over Network Schemas," Proc. of the ACM SIGMOD, 1979, pp. 179-190.

付記　ＴＣＣＰ実現における状態遷移図

トランスポート群制御プロトコル（ＴＣＣＰ）の実現で用いられた状態遷移図を示す。

1. 記　　法



X：上位へorから　／X：送信
X：下位へorから　X／：受信

S　　：コマンドのＤＤＢフレーム
〈S〉　：データのＤＤＢフレーム（ストリーム）
G　　：コマンドのＴＣＣフレーム
〈G〉　：データのＴＣＣフレーム（セグメント）
TO(GTO)：タイマー（グローバルタイマ）のタイムアウト
〈G〉⫽　：全Ｔ　ＣＥからのセグメントＧの受信

$\left\{\begin{array}{l}\langle G_1\rangle\\ \vdots\\ \langle Gm\rangle\end{array}\right\}$／：$G_1$，…，$Gm$ のうちのいずれかのセグメントの受信

$\left\{\begin{array}{l}\langle G_1\rangle\\ \vdots\\ \langle Gm\rangle\end{array}\right\}$⫽：$G_1$，…，$Gm$ のうちいずれかのセグメントを全ＴＣＣＥから受信

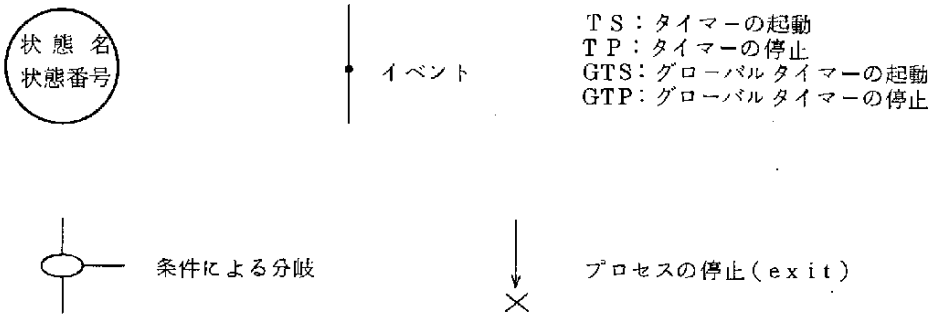
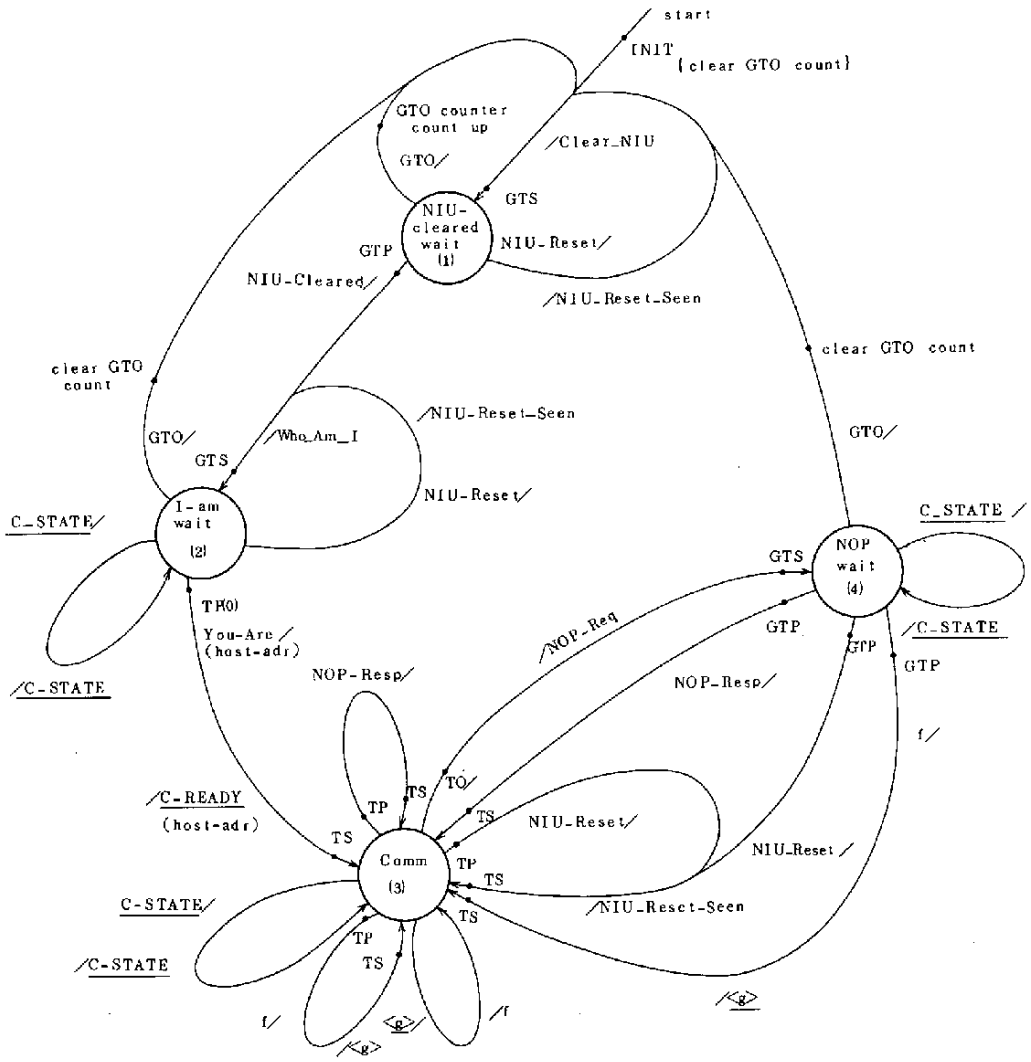


図１　状態遷移図の記法

2. DLE



INIT : DLEinitialize

図2　ＤＬＥ状態遷移図（1/3）

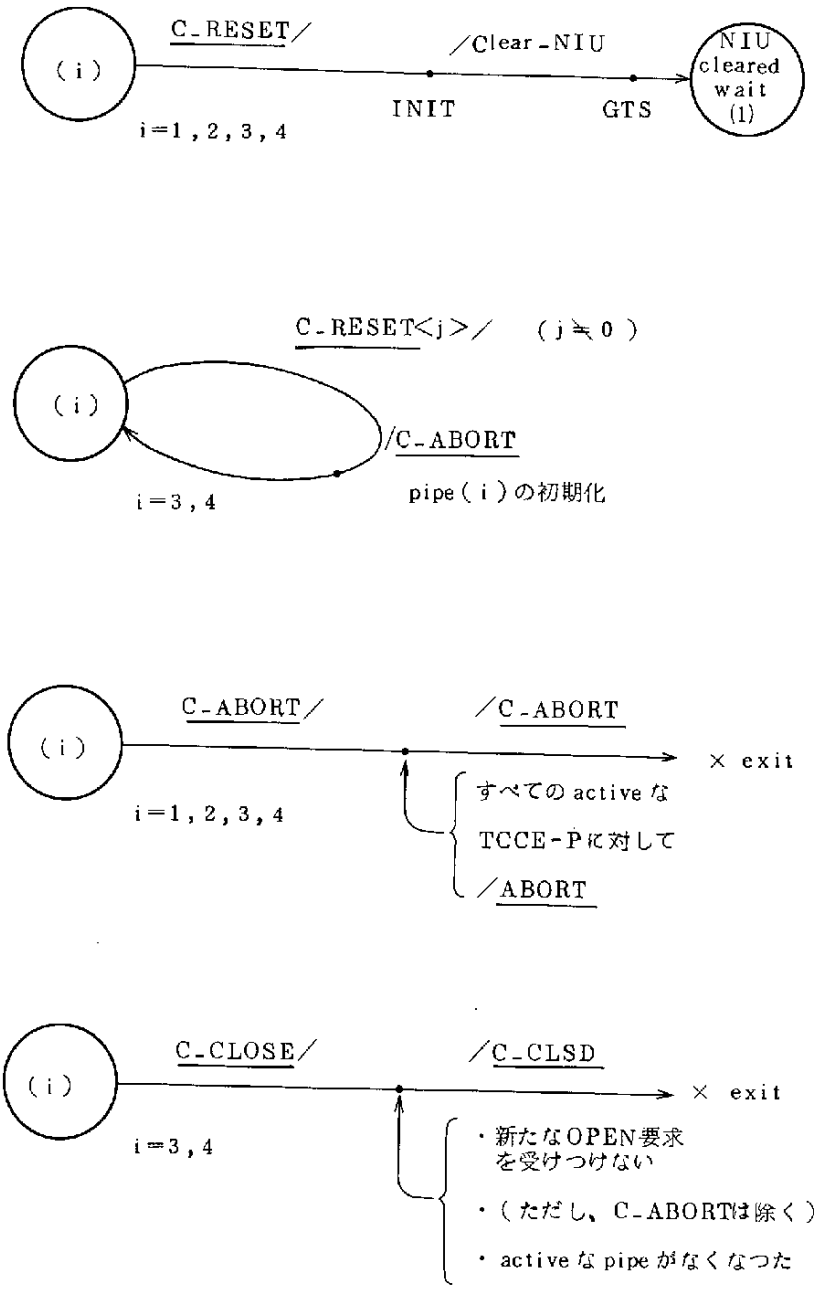
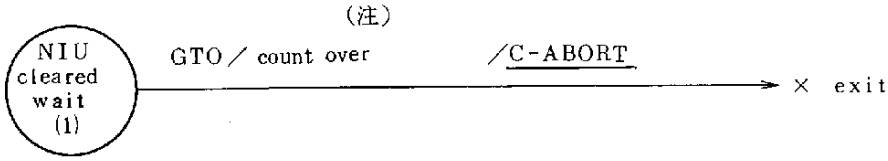


図２　ＤＬＥ状態遷移図(2/3)



（注）NIU_cleared_waitのとき、ＧＴＯ に対する
Ｔimeoutが発生したとき、そのＴimeoutの
回数をcountする。

Ｔimeout回数が TMMAX を超えたとき、
ＧＴＯ count over.

TMMAX：Ｔimeout回数
の制限値。

図２　ＤＬＥ状態遷移図(3／3)

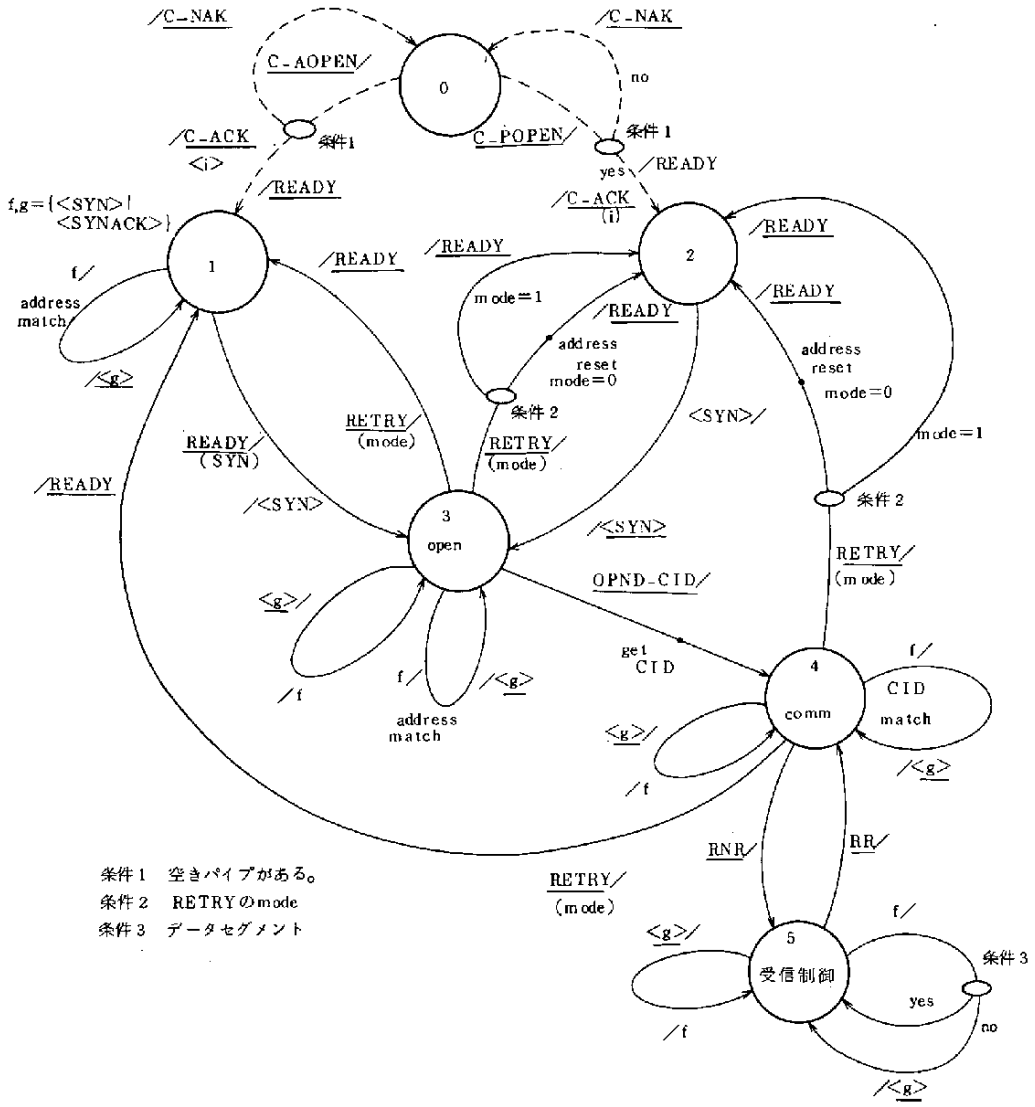
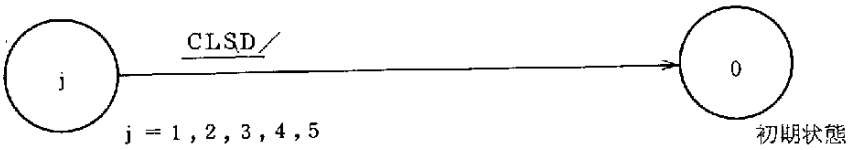


図3 パイプiの状態遷移図(1/2)



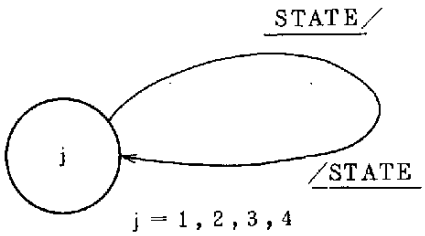


図３　パイプｉの状態遷移図（2／2）

3. TCES

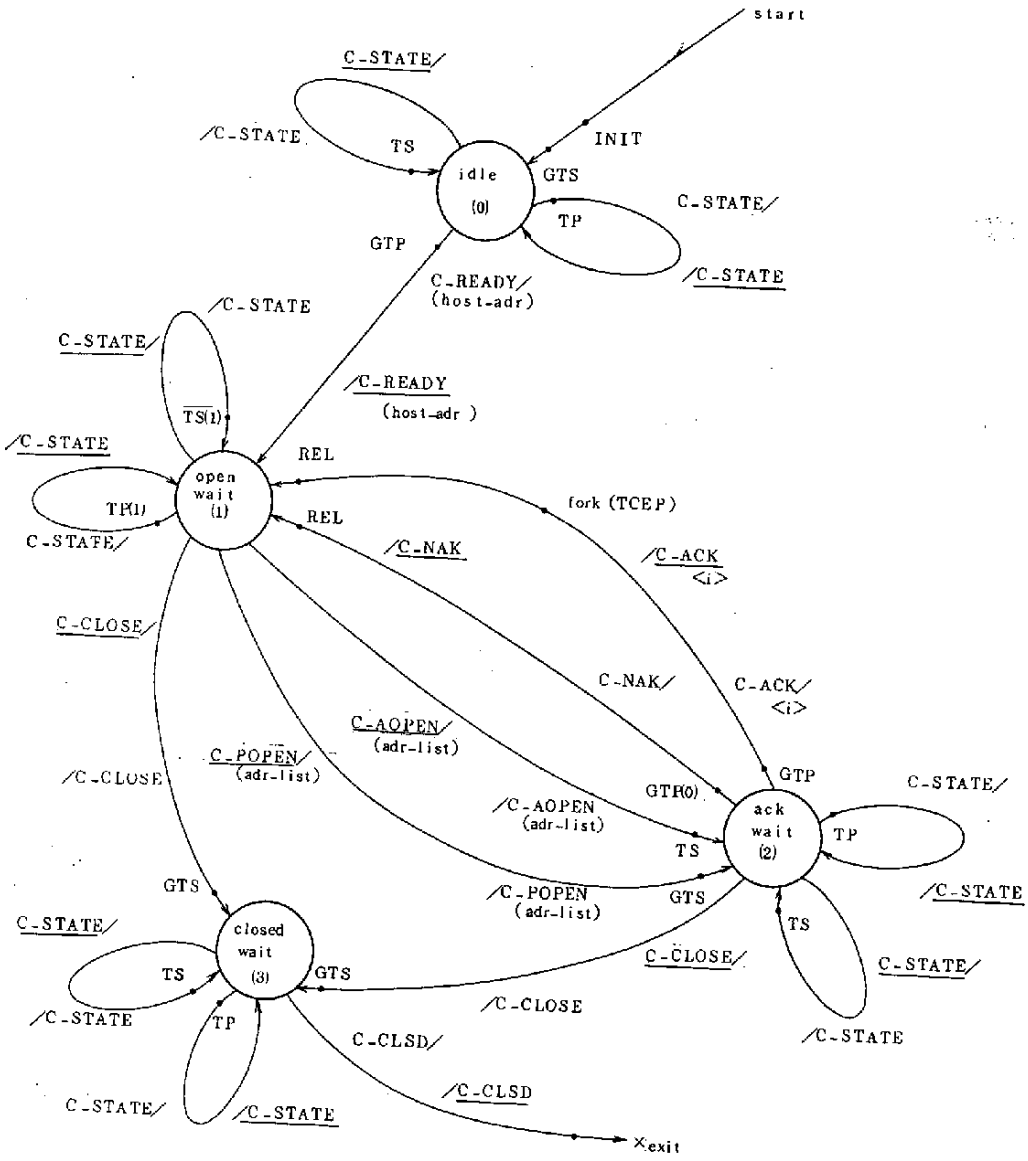
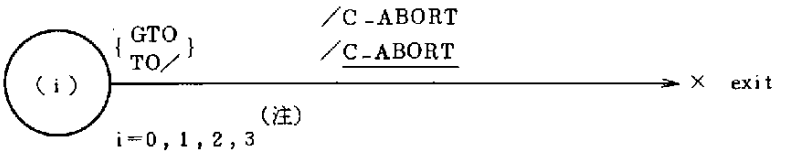
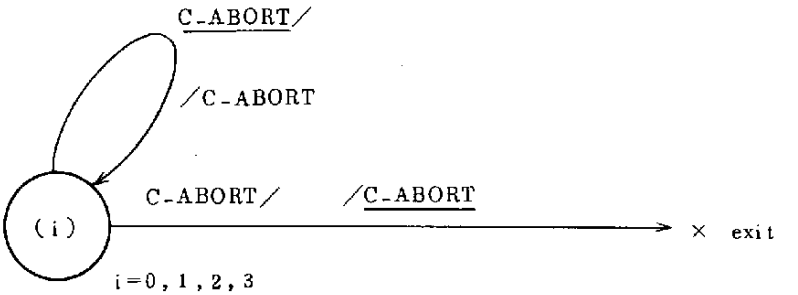


図 4　TCCE－S 状態遷移図(1)



(注) i = 1のときGTOに対する
Timeout は除く



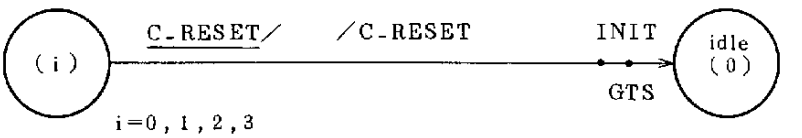


図4　TCCE－S状態遷移図(2)

4. TCEP

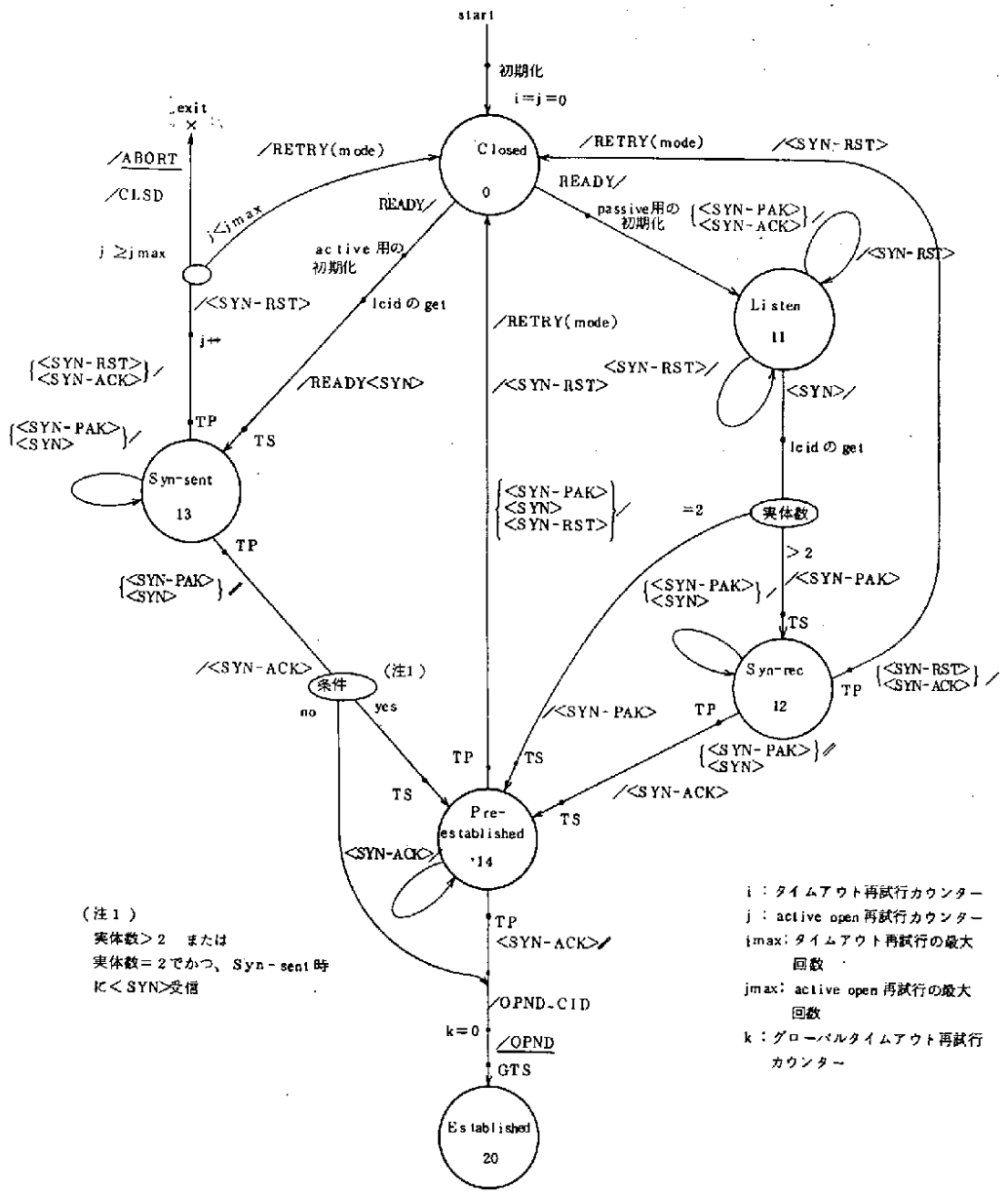


図 5　群開設処理状態遷移図(1)

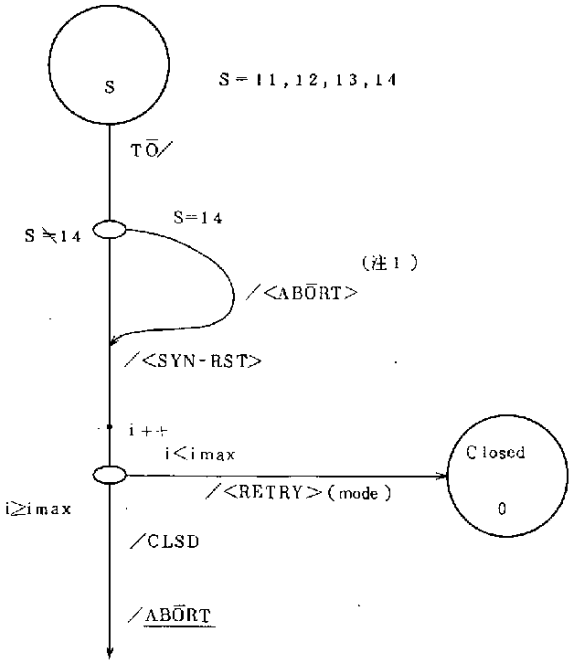
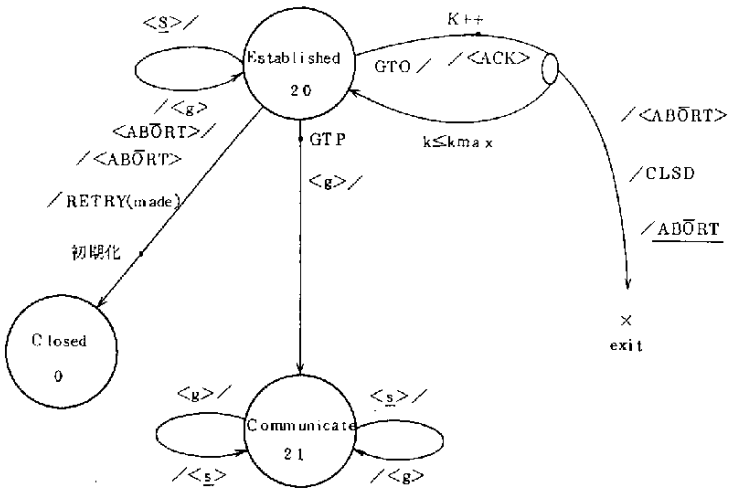
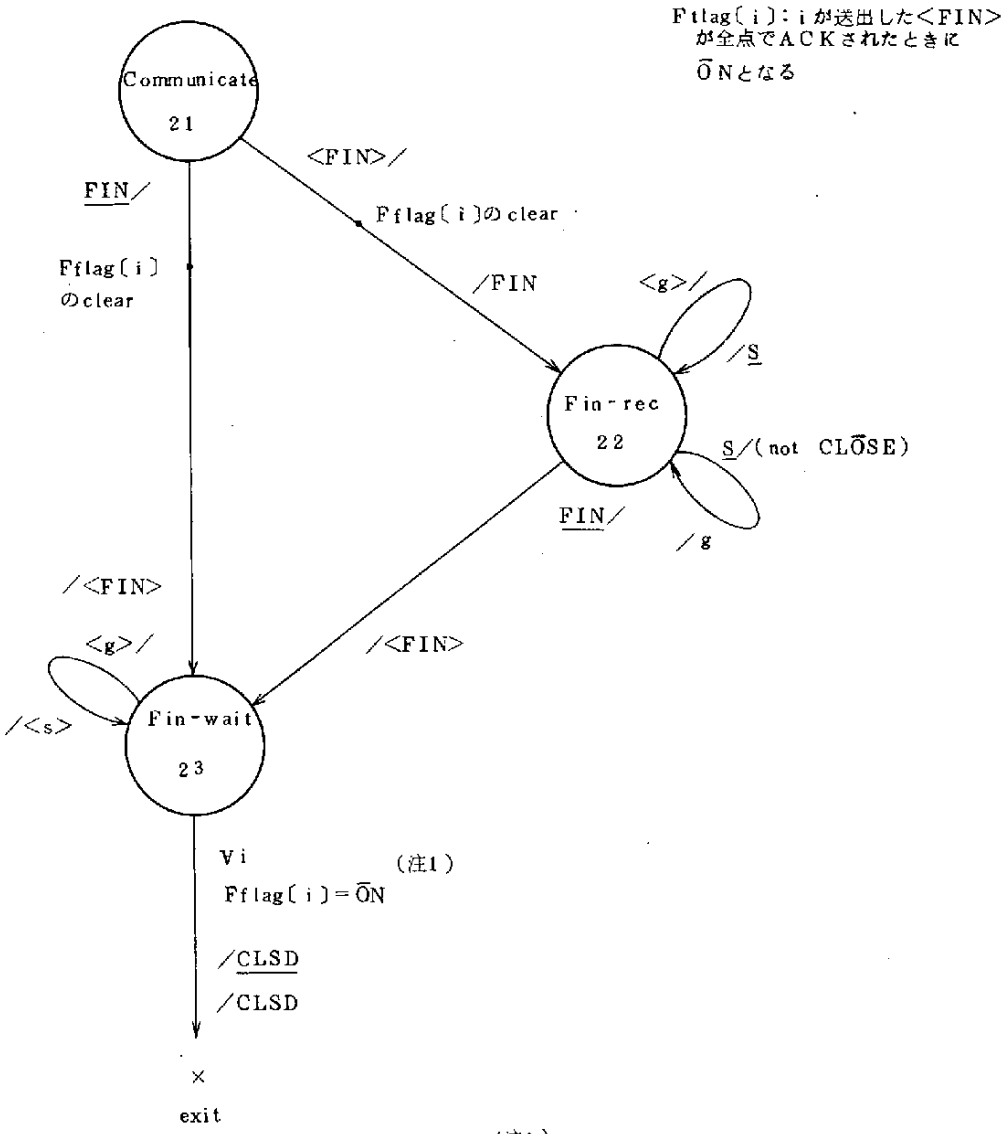


図5　群開設処理状態遷移図(2)



k　：グローバルタイムアウト再試行カウンター
kmax：　〃　　の最大試行回数

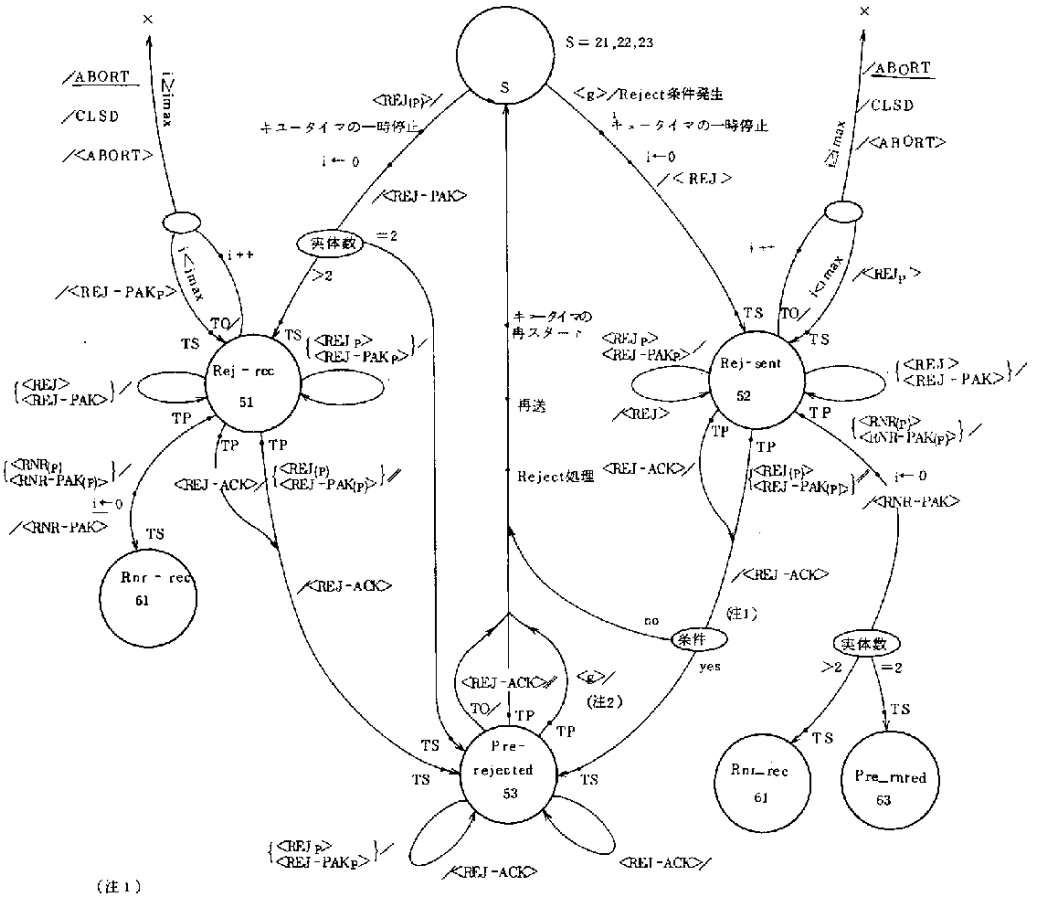
図6　データ転送処理状態遷移図



Fflag〔i〕：iが送出した<FIN>が全点でACKされたときにONとなる

（注1）
自分の送信した<FIN>がRTQ2よりdequeueされたときFIB〔i〕=ON（i＝自分）
受信した<FIN>がAQ2よりdequeueされたときFIB〔i〕=ON（i＝g．ENO）

図7　群解除処理状態遷移図



(注1)
実体数＞2、または
実体数＝2でかつRej-sent時
に<REJ(P)>を受信

(注2)
Reject処理後の通番エラーのない
セグメントを受信した場合

i：タイムアウト再試行カウンタ
imax　〃　の最大回数

図8　ＲＥＪ処理状態遷移図

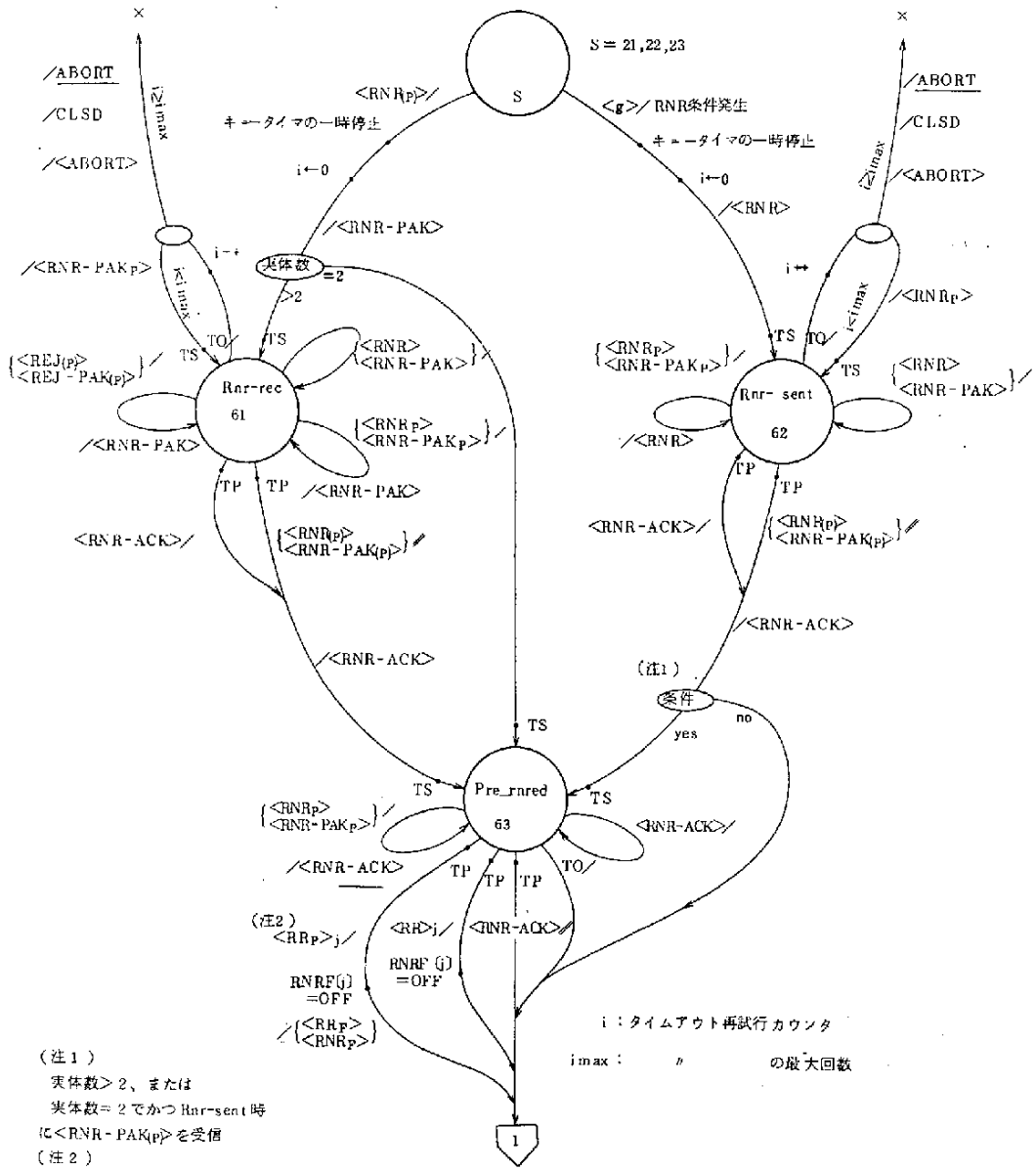


図9　フロー制御処理状態遷移図(1/2)

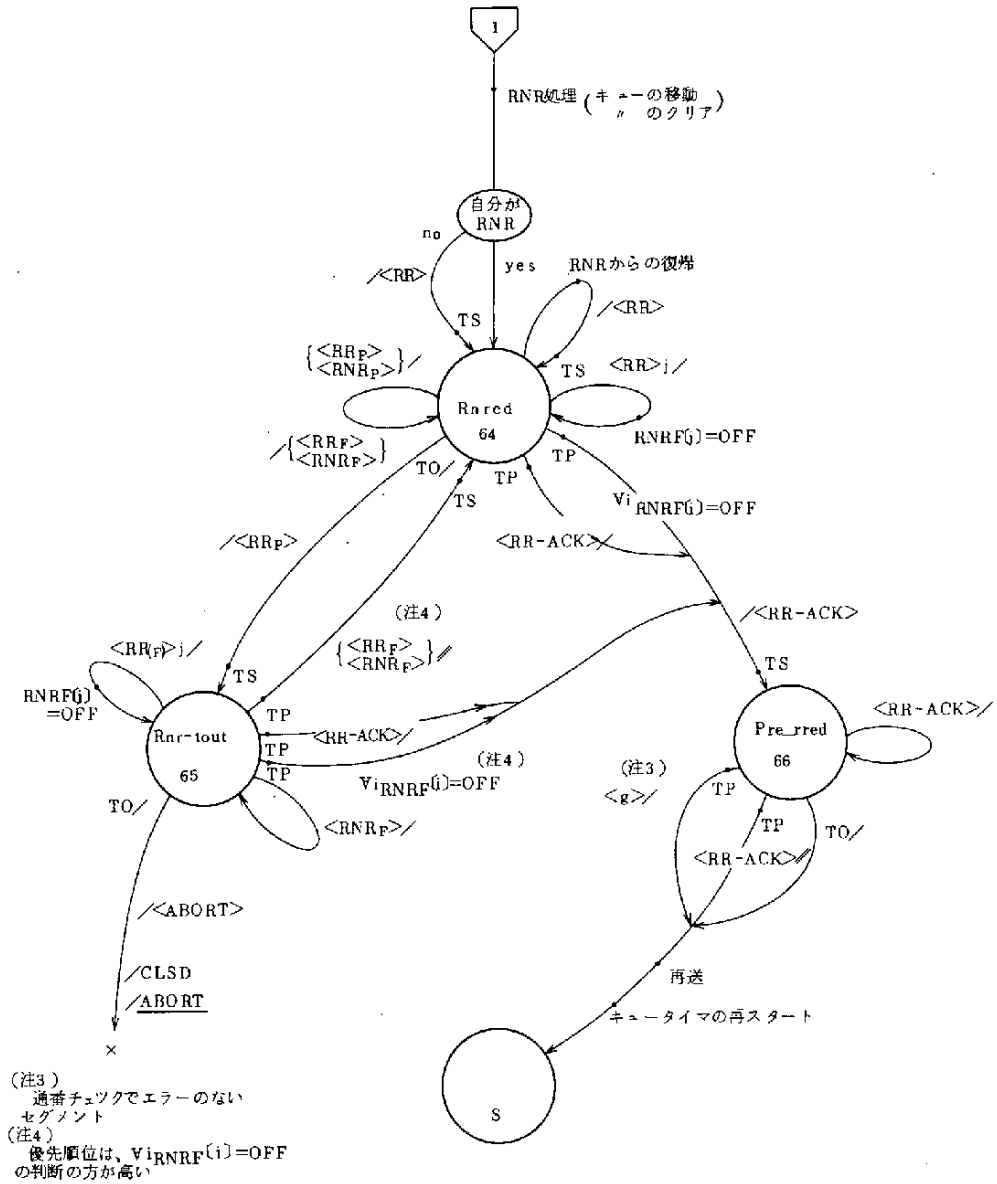


図 9　フロー制御処理状態遷移図 (2/2)

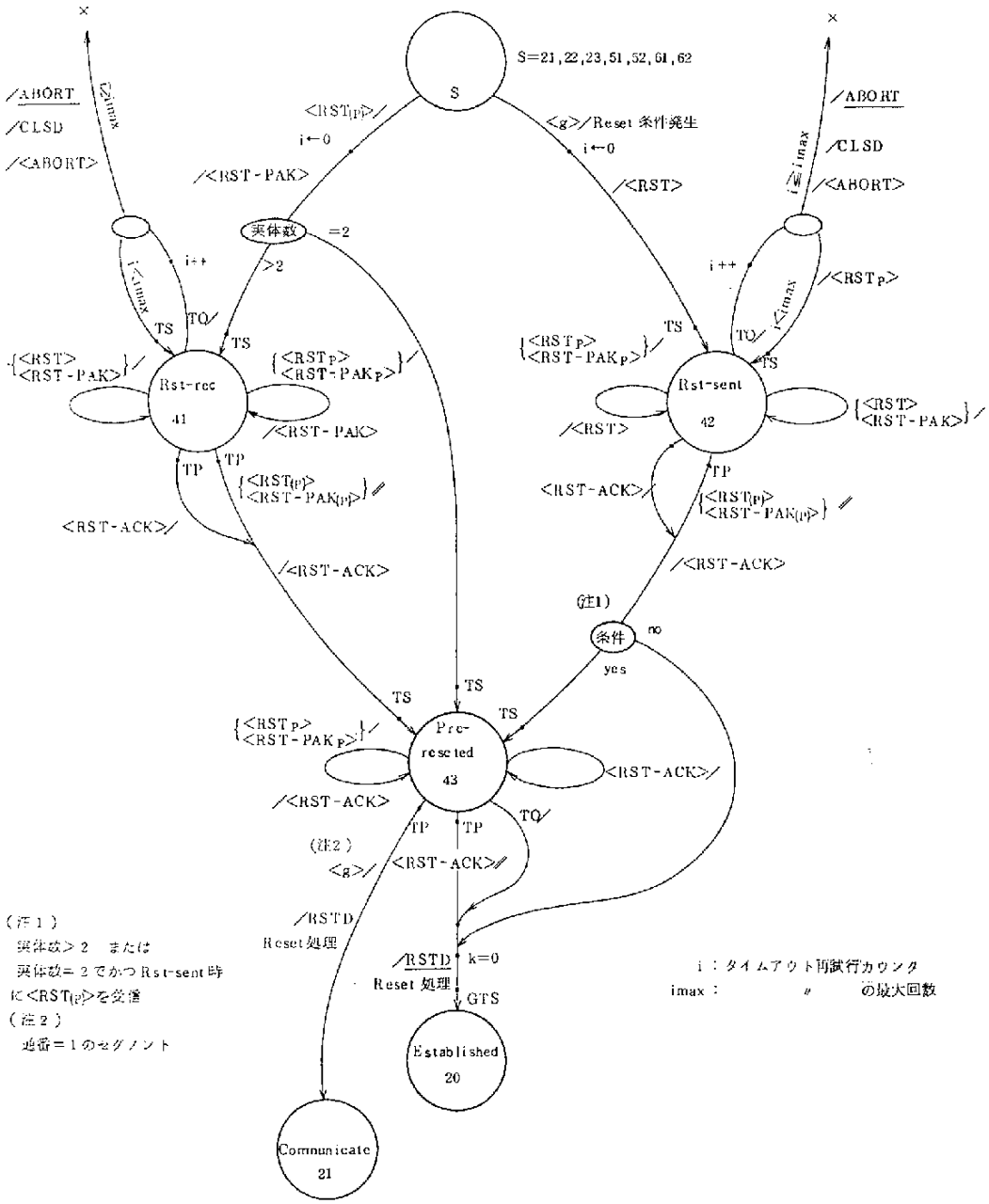


図１０　ＲＳＴ処理状態遷移図

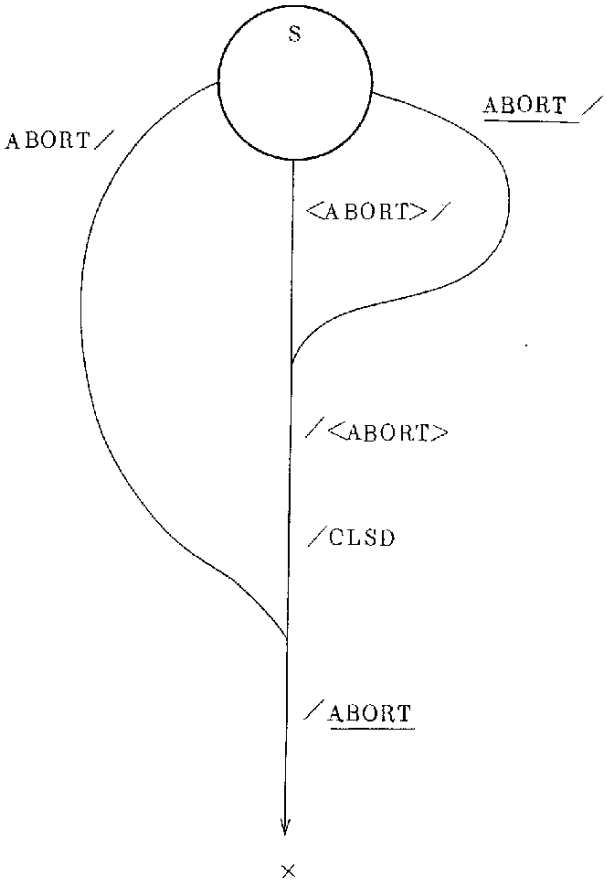


図１１　ＡＢＯＲＴ処理状態遷移図

付記　　記　　法

```
<xx>                        ; Segment を示す。
IN,OUT                      ; Segment の受信及び送信を示す。 (IN=<>,OUT=<>)
SEND xx to yy               ; コマンド xx を実体 yy に送る。
RECEIVE xx from yy          ; コマンド xx を実体 yy から受け取る。
TS                          ; Timer start
TP                          ; Timer stop
GTS                         ; Global Timer state
GTP                         ; Global Timer stop
QTS                         ; Queue Timer start
QTP                         ; Queue Timer stop

EN                          ; 群に加入している実体数。
ENO                         ; Segmentを送信した実体の群内の一連番号
Myno                        ; 自分の群内での一連番号

Table                       ; SYN,REJ,RNR,RR,RSTのテーブルが有り、各テーブルはEN個の領域からなる。
RR Table all ON             ; 総ての領域を ON (1) する。
XXX Table all ON            ; XXX Table (Myno) のみ OFF (0) その他の領域を総て ON する。

Ignore                      ; 受信 Segmentを無視する。
EXIT                        ; Process を総て終了する。
Retransmitte                ; 再送キュー内の Segmentの再送。
Queue Process               ; 各キューの処理。
Seg. Input Process          ; Segment の受信処理。
Seg. Output Process         ; Segment の送信処理。
Process of Data transfer    ; データの送受信に伴う全般の処理。
```

表1 群開設処理

	<SYN>	<SYN-PAK>	<SYN-ACK>	<SYN-RST>	Time Out
Listen 1 1	Get lcid if Not define Address list Set EN,Myno,Address list OUT=<SYN-PAK > SYN Table (ENO) OFF if EN>2 TS ; STATE=Syn-rec else TS ; STATE=Pre-established	OUT=<SYN-RST >	OUT=<SYN-RST >	Ignore	
Syn-rec 1 2	if Matched lcid list SYN Table (ENO) OFF if SYN Table all OFF TP OUT=<SYN-ACK > SYN Table all ON TS STATE=Pre-estabilised else Reset Output Process	if Matched lcid list SYN Table (ENO) OFF if SYN Table all OFF TP OUT=<SYN-ACK > SYN Table all ON TS STATE=Pre-estabilised else Reset Output Process	if Matched lcid list TP OUT=<SYN-RST > SEND RETRY (mode) to DLE STATE=Closed else Reset Output Process	if Matched lcid list TP OUT=<SYN-RST > SEND RETRY (mode) to DLE STATE=Closed else Reset Output Process	Time out process
Syn-sent 1 3	if Matched lcid list SYN Table (ENO) OFF if SYN Table all OFF TP OUT =<SYN-ACK > SYN Table all ON TS STATE =Pre-established else Reset Output Process	if Matched lcid list SYN Table (ENO) OFF if SYN Table all OFF TP ; OUT =<SYN-ACK > if EN >2 SYN Table all ON TS ; STATE =Pre-established else SEND OPEND-CID to DLE SEND OPND to DDBE k =0 ; GTS STATE =Established else Reset Output Process	if Matched lcid list TP OUT=<SYN-RST > j++ if j >jmax SEND CLSD to DLE SEND ABORT to DDBE EXIT else SEND RETRY (mode) to DLE else Reset Output Process	if Matched lcid list TP OUT=<SYN-RST > j++ if j >jmax SEND CLSD to DLE SEND ABORT to DDBE EXIT else SEND RETRY (mode) to DLE else Reset Output Process	Time out process
Pre-established 1 4	if Matched lcid list TP OUT=<SYN-RST > STATE=Closed else Reset Output Process	if Matched lcid list TP OUT=<SYN-RST > STATE=Closed else Reset Output Process	if Matched lcid list SYN Table (ENO) OFF if SYN Table all OFF TP SEND OPEND-CID to DLE SEND OPEND to DDBE k=0 ; GTS STATE=Established else Reset Output Process	if Matched lcid list TP OUT=<SYN-RST > STATE=Closed else Reset Output Process	Time out process

表2 データ転送処理

	<g>	< RST>or< REJ>or< RNR>	<ABORT >	その他	Time Out
Established 20	GTP if < ACK> (*1) OUT=<ACK > else Seg. Input Process STATE=Communicate		GTP OUT =<ABORT > SEND RETRY (mode) to DLE STATE =Closed	RECEIVE Data Stream from DDBE Seg. Output Process	k++ if k >kmax OUT=<ABORT > SEND CLSD to DLE SEND ABORT to DDBE EXIT else OUT=<ACK > (*2) GTS
Communicate 21	if RNR Condition RNR Process else if Matched ACK Seg. Input Process else Reject Condition if Reset Condition RST Process else if Reset Condition RST Process	if IN =<RST > STATE =Rst-rec if IN =<REJ > STATE =Rej-rec if IN =<RNR > STATE =Rej-rec	RNR Process	RECEIVE Data Stream from DDBE Seg. Output Process	Queue Process

注) *1 ack (ENO) =1 かつ ack (n) =0 (n ≒ENO) である< ACK>
*2 ack (Myno) =1 かつ ack (n) =0 (n ≒ENO) である< ACK>

表3 各 Proccs

Closed Process	Time Out Process	ABORT Process	Reset Output Process
i =0 ; j =0 closed1 if A-open *3 Set EN,Myno,Addres-list Get lcid OUT =< SYN> TS STATE =Syn-sent else if Address wer Defined Set EN,Myno,Address-list STAT=Listen	if STATE=Pre-established OUT =< ABORT> OUT =<Syn-rest> i++ if i<imax SEND RETRY (mode) to DLE STATEA=Closed else SEND CLSD to DLE SEND ABORT to DDBE EXIT	OUT =<ABORT > SEND CLOSE to DLE SEND ABORT to DDBE	if All lcid are not equal lcid list =Input Segment's lcid list else lcid list =My lcid list OUT =<SYN-RST >

注) *3 14,20 において Closed に戻る場合 closed1に戻る。

表4 REJ 処理

	<REJ>	<REJ-PAK>	<REJ-ACK>	<g>	<RNR >, <RNR-PAK >	TIME OUT
STATE = 21,22,23	REJ Table all ON QTP ; i =0 OLD-State=STATE OUT=<REJ-PAK > if EN>2 STATE=Rej-rec TS else STATE=Pre-rejected TS	Ignore	Ignore	if Reject condition QTP ; i =0 OLD-state =STATE REJ Table all ON OUT =<RST > TS STATE =Rej-sent else Process of Data transfer		
Rej-rec 5 1	REJ Table (ENO) OFF if P bit =1 OUT=<REJ-PAK > if REJ Table all OFF TP REJ Table all ON OUT=<RJE-ACK > TS STATE=Pre-rejected	REJ Table (ENO) OFF if P bit =1 OUT=<REJ-PAK > if REJ Table all OFF TP REJ Table all ON OUT=<REJ-ACK > TS STATE=Pre-rejected	TP REJ Table all ON OUT=<REJ-ACK > TS STATE=Pre-rejected	if IN=<RST > or <RST-PAK > TP RST Table all ON OUT=<RST-PAK > i=0 ; TS STATE=Rst-rec else Ignore	TP i=0 RNR Table all ON OUT=<RNR-PAK > TS STATE=Rnr-rec	i++ if i <imax OUT=<REJ-PAKp> TS else ABORT PROCESS
Rej-sent 5 2	REJ Table (ENO) OFF if P bit =1 OUT=<REJ > if REJ Table all OFF TP REJ Table all ON OUT=<REJ-ACK > TS STATE=Pre-rejected	REJ Table (ENO) OFF if P bit =1 OUT=<REJ > if REJ Table all OFF TP REJ Table all ON OUT=<REJ-ACK > if EN>2 TS STATE=Pre-rejected else Reject Process QTS STATE=OLD-state	TP OUT=<REJ-ACK > if EN>2 REJ Table all ON TS STATE=Pre-rejected else Reject Process QTS STATE=OLD-state	if IN=<RST > or <RST-PAK > TP RST Table all ON OUT=<RST-PAK > if EN>2 i=0 ; TS STATE=Rst-rec else TS STATE=Pre-reseted else Ignore	TP RNR Table all ON OUT=<RNR-PAK > if EN>2 i=0 ; TS STATE=Rnr-rec else TS STATE=Pre-rnred	i++ if i <imax OUT=<REJp> TS else ABORT PROCESS
pre-rejected 5 3	if P bit ON OUT=<REJ-ACK > else Ignore	if P bit ON OUT=<REJ-ACK > else Ignore	RJE Table (ENO) OFF if REJ Table all OFF TP Reject Process QTS STATE=OLD-state	if Matched ACK Seq. TP Reject Process QTS STATE=OLD-state else Ignore	Ignore	QTS STATE=OLD-state

表5 Flow Control 処理 (1)

	<RNR>	<RNR-PAK>	<RNR-ACK>	<g>	その他	TIME OUT
STATE = 21,22,23	RNR Table all ON QTP ; i=0 OLD-State=STATE OUT=<RNR-PAK > if EN>2 TS ; STATE=Rnr-rec else TS ; STATE=Pre-rnred	Ignore	Ignore	if Rnr condition QTP ; i=0 OLD-State=STATE RNR Table all ON OUT=<RNR > TS ; STATE=Rnr-sent else Process of Data transfer		
Rnr-rec 6 1	Rnr Table (ENQ) OFF if P bit =1 OUT=<RNR-PAK > if RNR Table all OFF TP RNR Table all ON OUT=<RNR-ACK > TS STATE=Pre-rnred	RNR Table (ENQ) OFF if P bit =1 OUT=<RNR-PAK > if RNR Table all OFF TP RNR Table all ON OUT=<RNR-ACK > TS STATE=Pre-rnred	TP RNR Table all ON OUT=<RNR-ACK > TS STATE=Pre-rnred	if IN=<RST > or <RST-PAK > TP ; RST Table all ON OUT =<RST-PAK > i =0 TP STATE =Rst-rec else Ignore	if IN=<REJ > or <REJ-PAK > OUT=<RNR-PAK > else Ignore	i++ if i <imax OUT=<RNR-PAKp> TS else ABORT Process
Rnr-sent 6 2	RNR Table (ENO) OFF if P bit =1 OUT=<RNR > if RNR Table all OFF TP RNR Table all ON OUT=<RNR-ACK > TS STATE=Pre-rnred	RNR Table (ENO) OFF if P bit =1 OUT=<RNR > if RNR Table all OFF TP ; RNR Table all ON OUT=<RNR-ACK > if EN>2 TS STATE=Pre-rnred else Queue Process RR Table all ON if I am RR OUT=<RR> RR Table (Myno) OFF ; TS STATE=Rnred	TP OUT=<RNR-ACK > if EN>2 RNR Table all ON TS STATE=Pre-rnred else RR Table all ON Queue ProcessS if I am RR OUT=<RR> RR Table (Myno) OFF TS STATE=Rnred	if IN=<RST > or <RST-PAK > TP ; RST Table all ON OUT =<RST-PAK > if EN >2 i =0 TS STATE =Rst-rec else TS STATE =Pre-rested else Ignore	if IN=<RR> TP ; RR Table all ON RR Table (ENO) =OFF if P=1 if I am RR OUT=<RRf > else OUT=<RNRf> Queue Process if I am RR OUT=<RR> RR Table (Myno) OFF TS STATE=Rnred	i++ if i <imax OUT=<RNRp> TS else ABORT Process
Pre-rnred 6 3	if P bit ON OUT=<RNR-ACK > else Ignore	if P bit ON OUT=<RNR-ACK > else Ignore	RNR Table (ENO) OFF if RNR Table all OFF RR Table all ON TP ; Queue Process if I am RR RR Table (Myno) OFF TS STATE=Rnred	Ignore		RR Table all ON Queue Process if I am RR OUT=<RR> TS ; RR Table (Myno) OFF STATE=Rnred

表5 Flow Control 処理 (2)

	<RR>	<RR-ACK>	<RNR>	<g>	その他	TIME OUT
Rnred 6 4	RR Table (ENO) =OFF if P bit =1 if I am RR OUT=<RRf > else OUT=<RRf > if RR Table all OFF TP ; RNR Table all ON OUT=<RR-ACK> ; TS STATE=Pre-rred	TP RNR Table all ON OUT=<RR-ACK> TS STATE=Pre-rred	if P bit =1 if I am RR OUT=<RRf > else OUT=<RNRp>	Ignore	(Return from RNR) RR Table (Myno) OFF if RR Table all OFF OUT=<RR-ACK> RNR Table all ON TS STATE=Pre-rred else TS	OUT=<RRp > RNR Table all ON TS STATE=Rnr-tout
Rnr-tout 6 5	RR Table (ENO) OFF if RR Table all OFF TP RNR Table all ON OUT=<RR-ACK> TS STATE=Pre-rred else if P bit ON OUT=<RRf > RNR Table (ENO) OFF if RNR Table all OFF TP ; TS STATE=Rnred	TS RNR Table all ON OUT=<RR-ACK> TS STATE=Pre-rred	if P bit =1 RNR Table (ENO) OFF if RNR Table all OFF TS STATE=Rnred else Ignore	Ignore		ABORT Process
Pre-rred 6 6	if P bit =0 OUT=<RR-ACK> else Ignore	RNR Table (ENO) OFF if RNR Table all OFF TP Retransmitte QTS STATE=OLD-state	Ignore	if Matched ACK TS Process of input Retransmitte QTS STATE=OLD-state else Ignore		TS Retransmitte QTS STATE=OLD-state

RR Table は all ON 操作のとき RR Table (Myno) も ON にする。
その他の Table は all ON 操作のとき Table (Myno) は OFF とする。

表6 RST 処 理

	<RST>	<RST-PAK>	<RST-ACK>	<g>	Time Out
STATE =21,22,23, 51,52,61,62	RST Table all ON i=0 OUT=<RST-PAK > if BN>2 STATE=Rst-rec TS else STATE=Pre-reseted ; TS	Ignore	Ignore	if Rest condition RST Table all ON i=0 OUT=<RST > TS STATE=Rst-sent else Process of Data transfer	
Rst-rec 4 1	RST Table (ENO) OFF if P bit =1 OUT=<RST-PAK > if RST Table all OFF TP RST Tablt all ON OUT=<RST-ACK > TS STATE=Pre-reseted.	RST Table (ENO) OFF if P bit =1 OUT=<RST-PAK > if RST Table all OFF TP RST Table all ON OUT=<RST-ACK > TS STATE=Pre-reseted	TP RST Table all ON OUT=<RST-ACK > TS STATE=Pre-reseted	if IN=<RNR > or <RNR-PAK > OUT=<RST-PAK > else if IN=<REJ > or <REJ-PAK > OUT=<RST-PAK > else Ignore	i++ if i <imax OUT=<RST-PAKp> TS else ABORT Process
Rst-sent 4 2	RST Table (ENO) OFF if P bit =1 OUT=<RST > if RST Table all OFF TP RST Table all ON OUT=<RST-ACK > TS STATE=Pre-reseted	RST Table (ENO) OFF if P bit =1 OUT=<RST > if RST Table all OFF TP RST Table all ON OUT=<RST-ACK > if EN>2 TS STATE=Pre-reseted else Reset Process SEND RSTD to DDBE k=0 ; GTS STATE=Established	TP OUT=<RST-ACK > if BN>2- RST Table all ON TS STATE=Pre-reseted else Reset Process SEND RSTD to DDBE k=0 ; GTS STATE=Established	if IN=<RNR > or <RNR-PAK > OUT=<RST > else if IN=<REJ > or <REJ-PAK > OUT=<RST > else Ignore	i++ if i >imax OUT=<RSTp> TS else ABORT Process
Pre-rested 4 3	if P bit ON OUT=<RST-ACK > else Ignore	if P bit ON OUT=<RST-ACK > else Ignore	RST Table (ENO) OFF if RST Table all OFF TP Reset Process SEND RSTD to DDBE k=0 GTS STATE=Established	if ACK (ENO) =1 TP Reset Process Send RSTD to DDBE k=0 ; GTS STATE=Established else Ignore	SEND RSTD to DDBE k=0 GTS STATE=Established

昭和 ６０年３月発行

発行所　財団法人　日本情報処理開発協会
東京都港区芝公園３－５－８
機 械 振 興 会 館 内
ＴＥＬ （４３４）８２１１（代表）

印刷所　株式会社　三 州 社
東京都港区芝大門１－１－２１
ＴＥＬ （４３３）１４８４