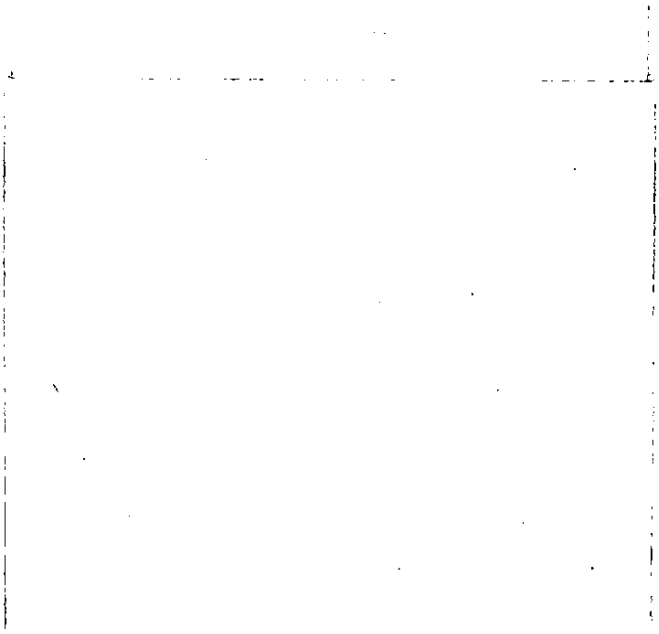
60－S 0 0 1

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 61 年 3 月

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60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ツール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，最終年度として，本事業全体の成果をとりまとめたものである。すなわち，ローカル情報システム（ＬＩＳ）の各プロトコルの実現，評価，パーソナル・コンピュータ，ワークステーション上の関係型ＤＢＭＳ（ＰＲＤＢＭＳ）の開発，改良，評価，高水準インタフェース（ＬＩＰ）の設計，開発，全体の統合化システムとしてのＪＤＤＢＳ（ＪＩＰＤＥＣ分散型データベースシステム）の諸概念について解説されている。

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６１年３月

目　　次

1. はじめに

情報化社会といわれる今日では，具体的な「もの」に対して，無形な「情報」といわれるものが，社会的な価値を高め，商品としての地位を固めてきている。こうした情報化社会にあって，各組織体，企業体では，自らが有しているあらゆる情報，例えば，経理，顧客，在庫，製品の設計，製造，保守についての情報，更に，専門家の知識，ノウハウといった情報を，組織体として統合的に管理することが求められてきている。むしろ，いかに情報を把握するかが，各組織体が情報化社会で生きのびる条件となっているといえる。

こうした情報化社会の進展のなかで，情報を記憶し，必要な情報を迅速に取り出すための情報システムを，計算機により，いかに実現していくかが，重要な課題となっている。こうした今後の情報システムを考えていくうえで，次の計算機技術を考える必要がある。

まず，第一に注目すべき点は，近年のＶＬＳＩを中心としたハードウェア技術の進歩により，現在の中～大型機並みの処理速度と記憶能力を有した超小型計算機が現われてきていることである。こうした計算機は，スーパーパソコンはワークステーションといわれるものである。例えば，昨年秋に発表されたインテルのＭＰＵ８０３８６は，３～４ＭＩＰＳの処理速度で　テラバイトの仮想記憶空間を有している。こうしたハードウェアから成る高性能ワークステーション（ＷＳ）は，ソウトウェア的にはＵＮＩＸといった標準的なＯＳを搭載してきている。各利用者は，こうした自分のＷＳを持つことにより，現在の中～大型機に匹敵する計算記憶能力を専有できるようになる。こうした高機能，高性能ＷＳは，今後の情報システムを考えていくうえで，最も重要な構成要素となっていくものと思われる。

次に重要な技術は，Ｅthernetに代表されるローカル・エリア網（ＬＡＮ）と，ＩＥＥＥとＩＳＯで進められている国際標準化である。ＬＡＮ，広域網（ＷＡＮ）といった通信網の進歩により，ＷＳ，大型機等の種々の計算機が，通信網により接続可能となるとともに，更に，通信網間，例えば，ＬＡＮとＷＡＮの接続が可能となってきている（図1.1）。今後の情報システムは，利用者のＷＳを中心に，大型機，超高速計算機等がＬＡＮ，ＷＡＮで結合された形態をとっていくものと考えられる。

さて，何故に，ＷＳ，大型機を，通信網に接続する必要があるのだろうか。こ

れは，計算機間での情報の交換のためである。情報は，計算機内では，データとして格納される。データを管理するシステムは，データベースシステム（ＤＢＳ）である。ＯＡ，ＣＡＤ，ＣＡＭを含めた種々の計算機応用は，ＤＢＳを核として構成されるとともに，ＷＳ間，ＷＳと大型機間での通信では，これらの計算機内のＤＢＳ内のデータが交換されることになる。

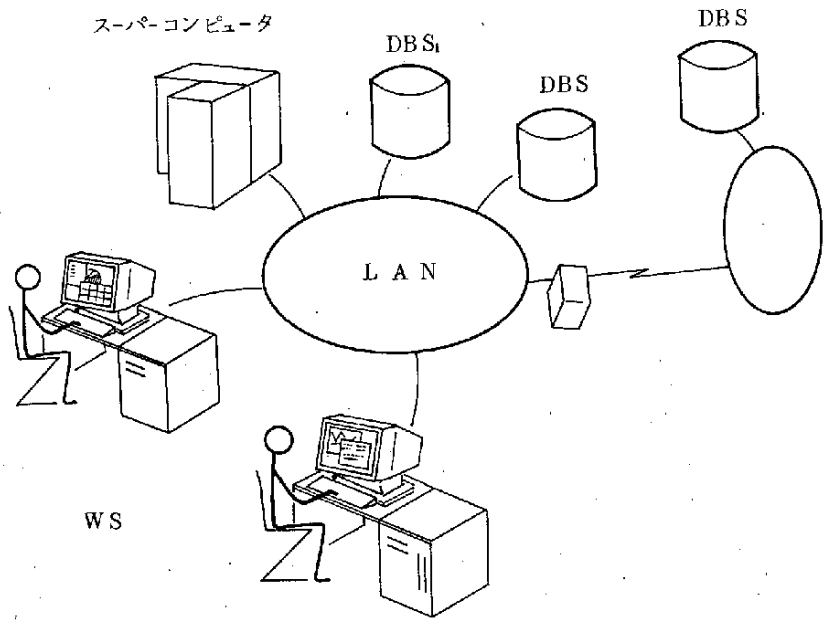


図 1.1 今後の情報システム

ＤＢＳが種々の計算機応用の核となってきているが，ＷＳと通信網の登場のなかで，ＤＢＳがどのような形態となっていくかが，本プロジェクトの主要な関心である。従来のＤＢＳは，一つの計算機内に存在し，複数の端末利用者によるデータの同時利用とインテグリティの保持を可能する集中型のシステムであった。これに対して，ＷＳと大型機が通信網で結合された情報システムでは，各計算機自身がＤＢＳを持ち，システム内でデータは各計算機に分散することになる。即ち，分散型のＤＢＳ，分散型ＤＢＳ（ＤＤＢＳ）となっていく。

こうして，ＤＢＳを中心に分散化されていく情報システムを実現するには，従来の集中型システムでは生じなかった次のような新たな技術的問題を解決していく必要がある。

(1) ＷＳ用のＤＢＳ。

(2) ＬＡＮの特長を生かした有効な通信プロトコル。

(3) 異種のＤＢＳの統合的利用方法。

(4) 分散したデータの安全性

本プロジェクトでは，以上の問題について取り組んできた。

本プロジェクトでは，図1.2に示すように，複数のWSと大型機が，LANにより接続された形態のもとで，各計算機のDBS間でのデータの共有を目指している。このシステムを，ローカル情報システム（LIS）という。

LISを実現するために，次の開発を行った。

(1) WS用のDBMS

(2) 高信頼放送通信プロトコル

(3) 高度利用者インタフェース

将来の分散システムにおいて，ワークステーション（WS）がこの基本要素となり，各WSでは，DBMS（データベース管理システム）を持つことが必須になる。しかしながら，現在のところ，パソコン，WS用の大～ミニコン並みの完全なDBMSは身近かにはない。このために，利用者にとって利用し易いリレーショナルDBMS（PRDBMS）の開発を行った。PRDBMSでは，例えば，

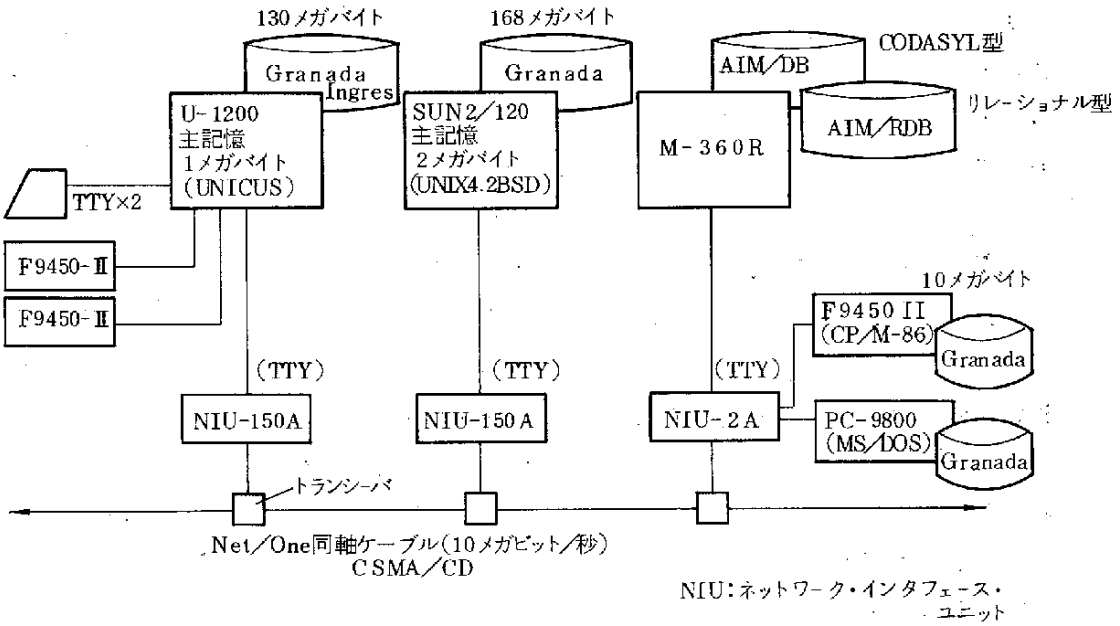


図1.2 LIPの構成

部を表すリレーション（表）ＤＥＰＴ（dno,dname）と，従業員を示すEMP（eno,name,age,dno）があると，これに対する検索「開発部員の中で３０才以下の社員名を求めよ」は，次のように書ける。

```
var （d,ＤＥＰＴ）　（c,ＥＭＰ）:
get〔e,name〕 d.dname＝" 開発 "　and
    d.dno＝e.dno　and　e.age＞＝３０;
```

ＰＲＤＢＭＳは，現在，Unix 4.2bsd ．及びＰＣ−９８００，ＩＢＭ５５５０のＭＳ／ＤＯＳ上で実働している。但し，５１２ＫＢの主記憶の必要である。

本ＰＲＤＢＭＳは，パソコンとＷＳでの本格的なＤＢＭＳとなるものであり，ＬＩＳの基本要素ともなるものである。

現在の通信網では，一対の実体（話者）間での一対一通信，即ち線通信が基本となっている。しかし，分散システムでは，複数の実体が同時に話合う" 面通信 "が必要になる。例えば，複数のＤＢＳに対する同時更新では，各ＤＢＳが互いに他のＤＢＳでの正常な更新が完了したことを確認する必要がある。このためには，高信頼な放送通信プロトコルが必要になる。高信頼とは，送信実体Ｓが，ｎ個の実体D_1，…，D_n にメッセージＭを送信したとき，ＳがD_1…，D_n が正しくＭを受信したことを確認するとともに，各D_i が他の全D_j が正しくＭを受信したことを確認できたとき初めて，ＭはD_1，…，D_n で受信されたことになる。このためのプロトコルとして，Ethernet 等のＬＡＮのＭＡＣ（媒体アクセス制御）層の放送通信サービスに，今述べた信頼性を付加したプロトコル（ＴＣＣＰＣトランスポート群制御プロトコル）を開発した。本プロトコルは，テレビ会議等の面通信を実現するうえでの基本通信を提供するものである。

本プロトコルは，現在Ｓun2／１２０上に，Unix 4.2 bsd を用いて実現されている。このプロトコルを用いて，ＤＢＳ間にまたがったトランザクションの分散処理の実験を行い，高信頼放送通信の有効性を明らかにできた。

ＬＩＳを実現するためには，各種のＤＢＳに対する共通なインタフェースが必要になる。このインタフェースとして，第５世代計算機の言語として検討されているＰＲＯＬＯＧを考え，既存のＣＯＤＡＳＹＬ型（ネットワーク型）のＤＢＳ上に，ＰＲＯＬＯＧインタフェースを設けた。このインタフェースをＬＩＰ（論理型言語インタフェース・プロセッサ）という。ＬＩＰは，ＣＯＤＡＳＹＬ型のＤＢＳだけでなく，我々の開発したリレーショナル型のＰＲＤＢＳ上にも容易に実現できるものである。以上のように，各ＤＢＳ上に，ＬＩＰを設けることによ

り，利用者は，ＤＢＳの差を意識することなく，同一の言語ＰＲＯＬＯＧでデータ操作が行えるとともに，従来のＤＢＳ言語にはない，演算規則を与えることによって，推論により，新しい事実を導出できる利点がある。

本プロダクトは，昭和５８年度に開始され，昭和６０年度に終了し，この中でＬＩＰの基本概念を定め，それを実現し，一部の実用化を行った。初年度は，基本概念を整理し，ＰＲＤＢＭＳのプロトタイプを開発した。第２年度には，ＬＡＮ，ワーステーション（ＷＳ）の導入が行われ，高信頼放送通信プロトコルの開発を行い，最終年度には，ＬＩＰとＬＩＳ全体の統合化を行った。

本プロジェクトで開発したＬＩＰは，今後の分散化を進める情報システムの基本アーキテクチャを与えるものであり，その一部は実用に供されている。ＷＳ用のPRDBMSは，既にパソコン（ＭＳ／ＤＯＳ）で実働し，他のパソコン用ＤＢＭＳよりも，完全なリレーショナルＤＢＭＳとしての機能を備えている。

本プロジェクトで考案した高信頼放送通信プロトコルは，従来の一対の実体間での線通信に対して，ｎ（２２）個の実体間での面通信を提供するものである。本方式は，今後のマルチメディア通信を含めた新たな通信システム設計の基本となっていくものである。

又，ＬＩＰは，従来のＤＢＳを事実の集合として，推論を行える知識ベースシステムでもある。現在の知識ベースシステムの多くが，主記憶内でのみ動作するものであるが，本システムは，既存のＤＢＳ上に推論機能を提供出来る利点をもつものである。

2. JDDBSの概要

JDDBS（JIPDEC Distributed Database System）は異種のDBS（Database System）を1つに統合化した分散データベースシステムである。

システムの構成としては，汎用DDBS及び，ローカル情報システム（LIS），高水準インタフェースシステム（LIP）の3つのサブシステムで構成される。LISはパーソナルコンピュータやワークステーション上のDBSを統合化したDDBSであり，個人ベースでの利用において有用なシステムとなっている。又，LIPは，DDBSの非手続き的なインタフェースであり，さらに強力な問い合せ処理，視野の拡大を可能としている。

JDDBSの初期に解決すべき問題として，

1. 各種DBSの異種性（2.2.1）
2. 各種DBSの分散性（2.2.2）

があり，これを解決するアプローチをまず示し，さらにLIS（2.3），LIP（2.4） 実現（2.5）について述べる。

2.1 全体アーキテクチャ

JDDBSの層構成による異種性，分散性の解決法及び全体のシステム構成について述べる。

2.1.1 4層構造

計算機科学においてよく用いられる手法として抽象化の概念がある。これは物理的特性やある特別な制約，特徴量といったものを，論理的又は概念的なレベルを設けることにより，見通しのよさ，わかり易さといったことを向上させるものである。JDDBSのアプローチも，各種DBSの異種性（データモデルの相違－後述）をあるレベルで同一化することや分散性（各DBSやDBの物理的位置）を解消するため，システム全体を4層に分けて抽象化を行なっている。

分散型データベースシステム（DDBS）とは種々のDBSを通信網によって結合し，利用者にあたかも1つのDBSかの様に見せるシステムである。このためには，上記の異種性，分散性を解決する必要がある。この2つの問

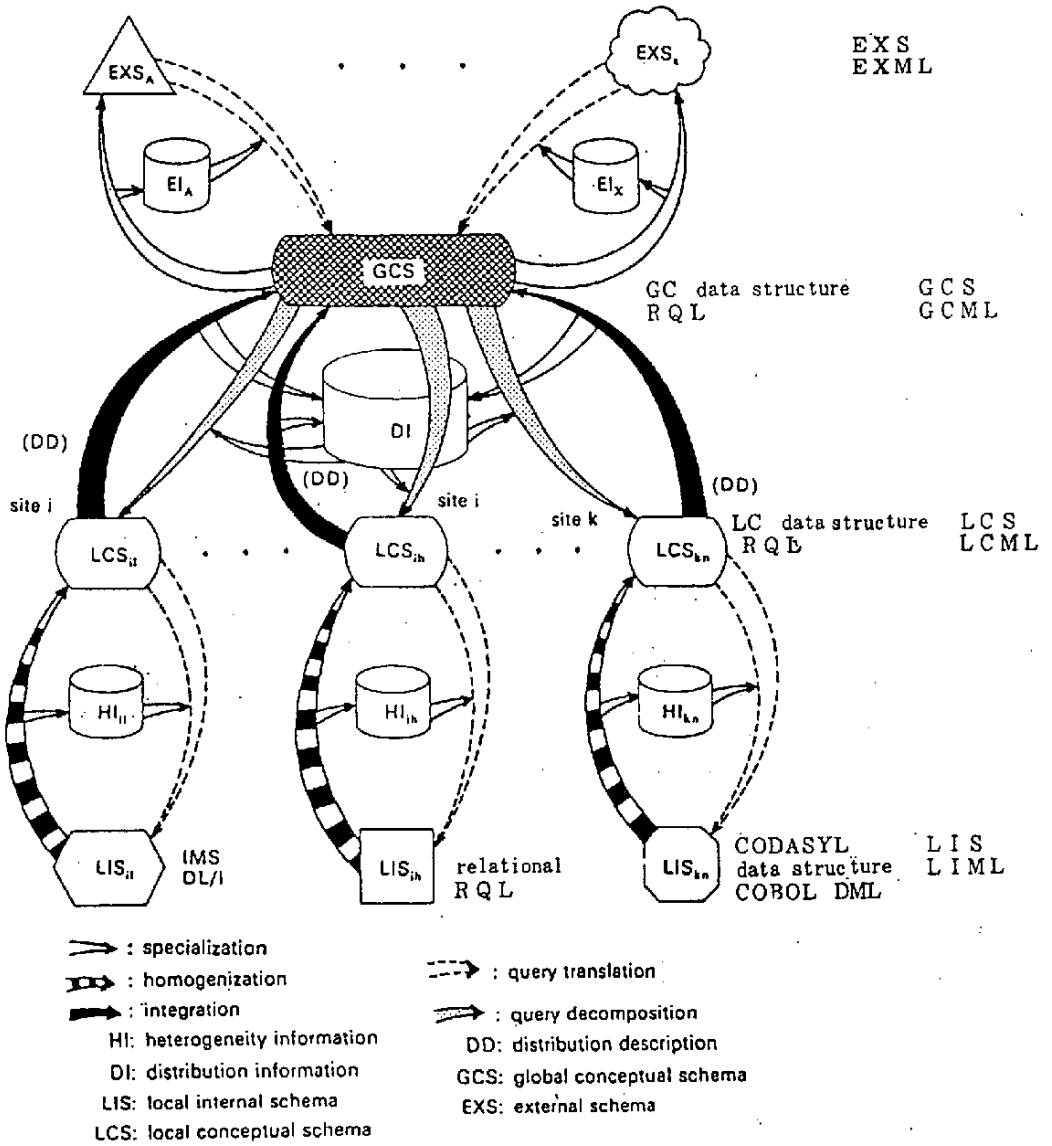


図 2.1　ＪＤＤＢＳの４層構造

題を解決するため，図2.1に示す様な4つの階層を考える。ローカル内部層は，既存ＤＢＳに対応し，ＤＢＳがＤＤＢＳを通して使用可能なデータ構造を表わすローカル内部スキーマ（ＬＩＳ）と操作言語（ＬＩＭＬ）を提供している。ＬＩＳとＬＩＭＬは，各ＤＢＳ固有のデータモデル（e.g. 関係モデル，CODASYLモデル）に基づいている。実際は，既存のＤＢＳはそれぞれ個有の制約や拡張機能を持っている。したがってこのレベルでは各既存のＤＢＳが採用しているデータモデルをある程度抽象化した形で定義しなおしている（仮想CODASYLモデル等）

ローカル概念（ＬＣ）層は，ＤＤＢＳ全体で共通なデータモデル（共通モデルとする）によるデータ構造を表すＬＣスキーマ（ＬＣＳ）とＬＣ操作言語（LCML）とから成る。ＬＣＳは，ＬＩＳを共通モデルにより表現したものである。この層では，利用者は各ＤＢＳの異種性（データモデルの差）を意識することなく，ＤＤＢＳ全体で共通なデータモデルのもとで，共通な言語ＬCMLを用いて，ＬＣデータ構造の操作が行える。つまり，この層で異種問題が解決されている。

全体概念（ＧＣ）層は，ＤＤＢＳ全体の一意なデータ構造記述としての全体概念（ＧＣ）スキーマ（ＧＣＳ），この上の操作言語（ＧCML）とから成る。ＧＣ層は，ＬＣ層と同一のデータモデルに基づいている。この層では，利用者は，各ＤＢＳの存在（即ち，分散性）を意識することなしに，DDBSを1つのＤＢＳのように見ることができる。この層では，異種性に加えて，分散性も解決されている。

外部（ＥＸ）層は，ＤＤＢＳの各応用固有のデータ構造としてのＥＸスキーマ（ＥＸＳ）とＥＸ操作言語（EXML）とから成る。

2.1.2 DDBSの構築

ＤＤＢＳの構築方法としては，既存ＤＢＳの存在を仮定して，構築する統合型設計と，1つの仮想ＤＢＳ（ＤＤＢＳ）を出発として，これを複数のＤＢＳで実現する分割型設計がある。

統合型設計では，各ＤＢＳのＬＩＳが前提として，これらの同種化を図るためＬＣＳが定義され，ＬＣＳの分散性を解決するため，ＧＣＳが定義される。最後に利用者個有のみかたを与えるためＥＸＳが定義される。これを特殊化という。

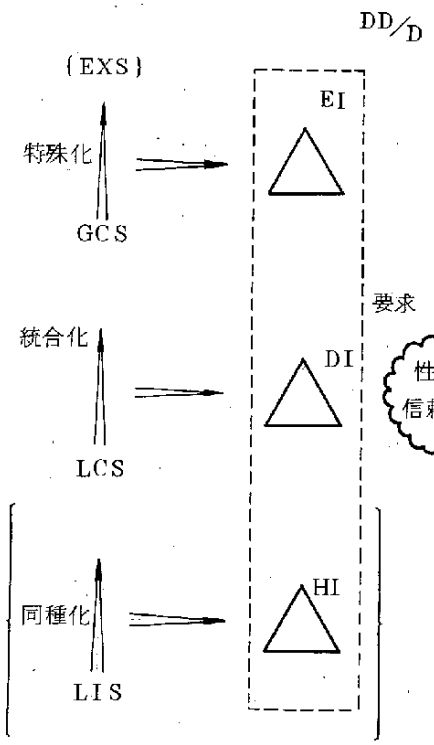


図 2.2　統合型設計

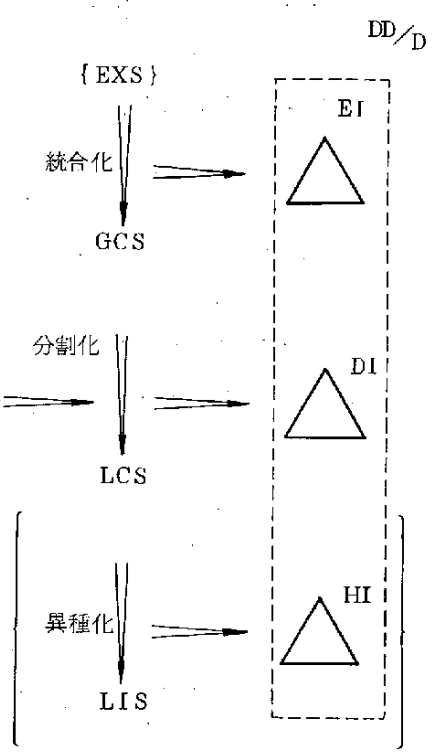


図 2.3　分割型設計

ＪDDBS は既存のＤＢＳが複数存在することを仮定しており，統合型設計により構築されている。

一方分割型設計では，パフォーマンス（応答性等），信頼性，データ獲得性，安全性等を考慮した分割が行なわれる。すなわちＥＸＳ，ＧＣＳが定義されたあと，上述した点を目標として，複数のＬＣＳに分割する。さらに，各ＤＢＳ固有のデータモデルでＬＣＳを表わしＬＩＳとする。

2.1.3　トランザクション管理

利用者がDDBSを利用する際に一つの処理単位となるのがトランザクションである。

ＪDDBSでは，利用者はまず，自分の応用毎の視野にもとずいて，ＥＸ層でＥＸＳに対する問合せ／更新演算から成る（ＥＸＭＬで記述）トランザクションT_EをDDBSに入力する。DDBSは，T_Eを全体概念（ＧＣ）層のＧＣＳ上のトランザクションT_Gに変換する。この変換は，外部情報ＥＩを用いて行われる。T_Gの参照するデータは，一般に，複数のＬＣＳ（即ち

異ったＤＢＳ）上に存在する。したがって，T_Gを各ＬＣＳ内のデータ構造に対するトランザクションT_{GL}に変換し，各ＤＢＳのローカル処理とＤＢＳ間での通信処理によって解を得る。これを分散トランザクション処理とよぶ。T_{GL}を，各ＬＣＳ内で処理出来る部分T_Lに分割することをトランザクション分割という。分散トランザクション処理は

(i) T_GからT_{GL}への変換

(ii) T_{GL}を各ＬＣＳのT_Lへ分割

(iii) ＤＢＳ間での通信処理

から成り，全ての処理は，ＤＩに基づいて行われる。さらに，T_Gの原子性とインテグリティを保つ為の同時実行制御，コミットメント制御，障害管理が必要になる。各ＤＢＳに分割されたトランザクションT_Lは，共通モデルに基づいているので，各ＤＢＳで実行可能なローカル内部（ＬＩ）操作言語（ＬIML）によるトランザクションT_Lに，異種性情報ＨＩを用いて変換され，ＤＢＳ上で実行され，ＤＤＢＳで共通なＬＣデータ構造に基づいた解が得られる。これをトランザクション変換とする。図2.4に概要を示す。

2.1.4 共通モデル

ＤＤＢＳの共通モデルとしては，いろいろな立場があるが，ここでは，種々のデータ構造を表現し得，かつ物理構造の依存性が低く，操作性に優れている点で，関係型のモデルを共通モデルとして採用している。

2.1.5 高水準インタフェース

ＪＤＤＢＳでは，ＥＸ層において，利用者の視野を，非決定的，再帰的に定義できるインタフェースとして論理型言語インタフェースプロセッサ（ＬＩＰ）を提供している。このインタフェースを用いることにより，従来，関係型モデルにおける操作において，不可能であった問い合せを行なうことが可能である。さらに，視野定義という点でも，仮想関係などが定義でき，より柔軟な構造となっている。

2.1.6 ローカル情報システム（ＬＩＳ）

ローカル情報システム（ＬＩＳ）はワークステーションやミニコン，パソコン等上の比較的小さなＤＢＳをローカルエリア網（ＬＡＮ）を用いて統合

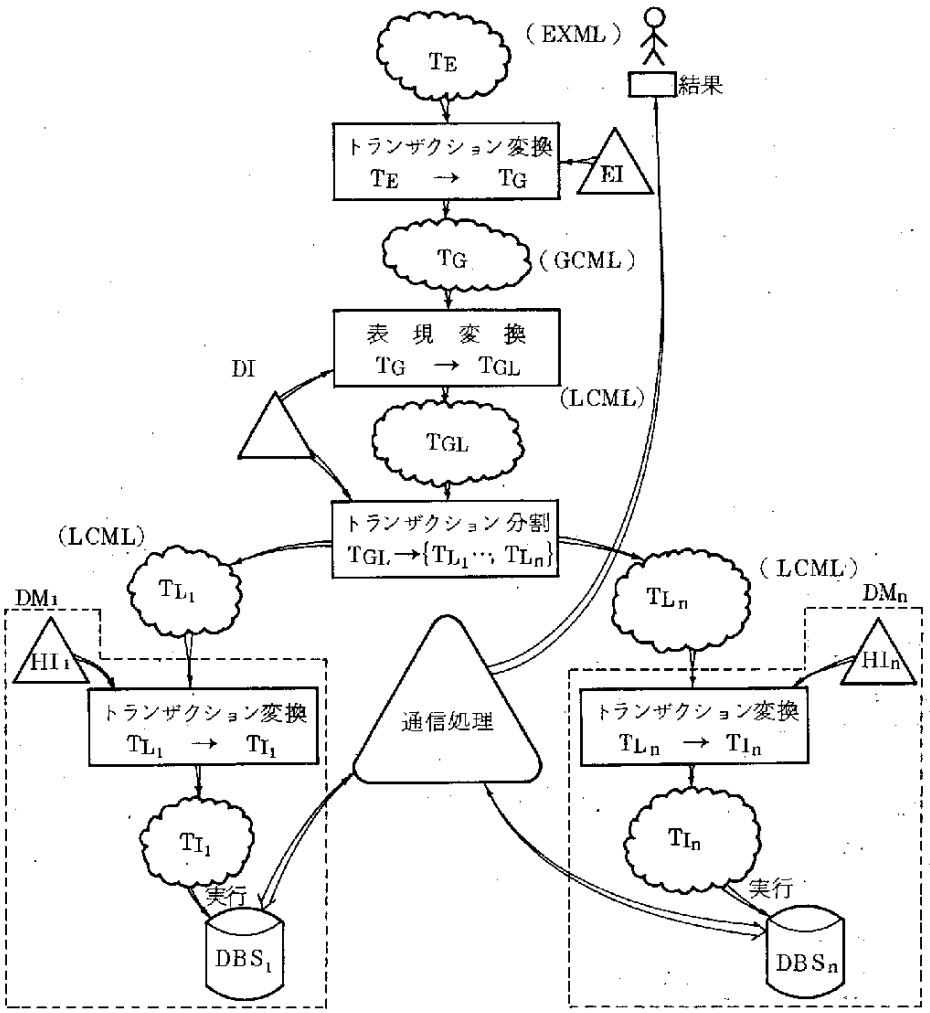
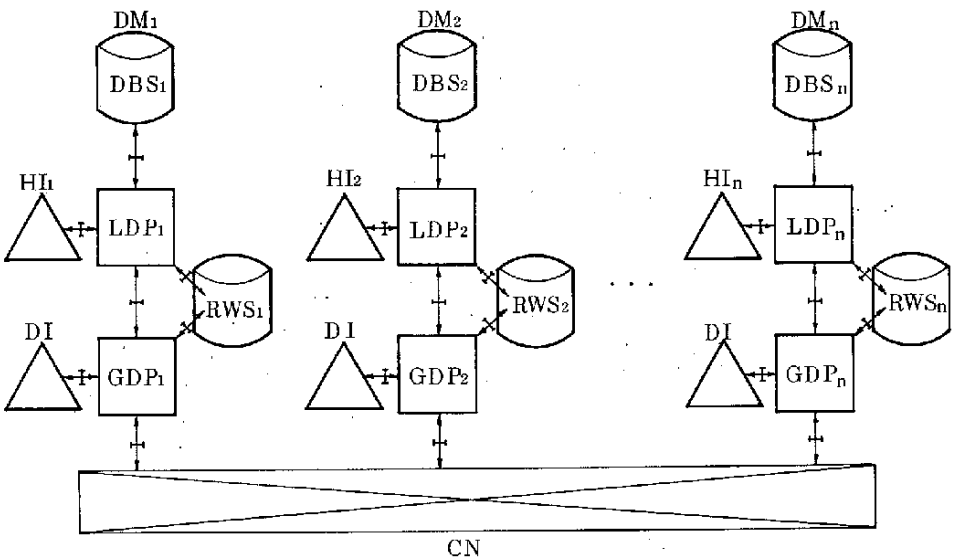


図 2.4　トランザクション処理

化したＤＤＢＳである。基本思想はＪＤＤＢＳと同じであり，複数の異種のＤＢＳを統合型設計を用いて，構築される。ＪＤＤＢＳ上では，ローカルなＤＤＢＳの一部として包含される。将来は，ＬＩＳを含めた，ＪＤＤＢＳの広範な統合化が考えられる。

2.1.7　JDDBSのシステム構成

四層構造に基づいたＤＤＢＳのシステム構成について論じる。ＤＢＭＳはｎ（２１）個のデータモジュール DM_1 ……，DM_n が通信網（ＣＮ）で結合されたシステム構成をとる（図2.5）。通信網は，データモジュール間の高信頼な種々の通信（１対１，１対多，多対多）をサポートする網で，広域ﾊ



HI: heterogeneity information
DI: distribution information
CN: communication network
DM: data module
DBS: database system
LDP: local database processor
GDP: global database processor
RWS: relational working storage

図2.5　ＪＤＤＢＳアーキテクチャ

ケット交換網，ローカルエリア網，バス，無線網等がこの例である。

各データモジュールDM_iは，次の6つのモジュール，全体データベースプロセッサ（GDP_i），ローカルデータベースプロセッサ（LDP_i），データベースシステム（DBS_i），異種性情報（HI_i），分散性情報（ＤＩ），関係作業域（RWS_i）から成る。データベースシステムDBS_iは，既存のＤＢＳであり，ローカル内部（ＬＩ）スキーマ（ＬＩＳ）に対して，ＬＩＳ操作言語（ＬＩＭＬ）で書かれたトランザクションを入力して，これを実行し，結果（問い合せならば導出データ）が得られるシステムである。ローカルデータベースプロセッサLDP_iは，既存のDBS_i上にあって，利用者にローカル概念（ＬＣ）層を提供するインタフェースである。LDP_iはスキーマ変換（同種化（ＬＩＳ→ＬＣＳ）／異種化（ＬＣＳ→ＬＩＳ）機能と，トランザクション変換（ＬＣＭＬ→ＬＩＭＬ）機能とを有している。前者では，ＬＩＳＬＣＳとの対応情報を異種性情報（HI_i）として生成し，後者では，このHI_iを用いてトランザクションの変換が行なわれる。トランザクション変換は，ＬＣＭＬトランザクションT_Lを，ＬＩＭＬトランザクションT_Lに変換し，これをDBS_jで実行させ，検索では，結果を共通な関係データ構造として，導出し，関係作業域RWS_iに関係として格納する。ＪＤＤＢＳのLDP_iは，(i)ＣＯＤＡＳＹＬ型のDBS_iのＣＯＤＡＳＹＬスキーマ（ＬＩＳ）から関係モデルのＬＣスキーマ（ＬＣＳ）の生成と，(ii)ＬＣＳ上のＲＱＬ問合せ（ＬＣＭＬ）をＣＯＢＯＬ　ＤＭＬ（ＬＩＭＬ）に変換し，実行し，解を関係としてRWS_iに格納する。

全体データベースプロセッサGDP_iは，LDP_i上にあって，利用者にＤＤＢＳ全体概念（ＧＣ）層を提供するプロセッサである。GDP_iの機能としては，統合型設計における統合化（｛ＬＣＳ｝→ＧＣＳ）と分割型設計における分割化（ＧＣＳ→｛ＬＣＳ｝），及び分散トランザクション処理の機能を有している。統合型設計では，ＤＤＢＳの全体管理者（ＧＡ）の定義したＧＣＳ定義から分散情報（ＧＩ）を生成し，ＤＤＢＳの全てのDM_jにその完全コピーを送る。分割型設計でも，ＤＩを生成し，他のDM_jに送る。

2.2　同種化と統合化

ここでは，2.1で述べた異種性と分散性を解決する手法を述べ，ＤＤＢＳを利用する意味で，トランザクション処理の概要について述べる。

2.2.1　異種性問題

2.1で述べたようにＤＢＳの異種性とは，そのＤＢＳが採用しているデータモデルの相違とする。データベースシステム（ＤＢＳ）は，利用者に対して，あるデータモデルに基づいたデータ構造と，その上の演算を提供するものと考えられる。商用のＤＢＳとして採用しているデータモデルとしては，関係型，ＣＯＤＡＳＹＬ型，階層型などがある。ここでは，データモデルは，そのデータ構造とそれに対する操作及びインテグリティ条件により定義する。異種ＤＢＳの同種化には，まず各ＤＢＳ上にＤＤＢＳ全体の共通モデルに基づいた視野を設ける必要がある。この共通モデルを概念モデルと呼ぶ。概念モデルは，各モデルの（インテグリティ条件（制約条件）を含めた）データ構造を表せる必要がある。

階層，ＣＯＤＡＳＹＬモデルは，実体と関連の２つの概念を有している。構造上の制約は，関連の型として特徴づけられる。例えばＣＯＤＡＳＹＬモデルは，レコード型は実体，セット型は実体間の１：ｍ（ｍ≧０）関係を表わしている。したがって，概念モデルでは，実体と関連との２つの独立な概念を有している必要がある。ここではＣＯＤＡＳＹＬモデルをとりあげ，概念モデルの特性化によってこれを表わす問題を論じる。（これは５章でも改めて定義される。）

2.2.1.1　概念モデル

概念データ構造＄は，時間不変な概念スキーマＳと，Ｓに実際に値を与え時間可変な概念データベースＳとから成る（＄＝（$\underline{S}$，Ｓ））。概念スキーマ＄は，２種の関係スキーム，Ｅ関係スキームとＲ関係スキームから成る。

Ｅ関係スキーム

Ｅ関係スキーム$\underline{E}$は，属性集合$\Omega_E = \{ @E, a_1, \cdots\cdots, a_m \}$（$m \geq 0$）とインテグリティ条件$T_E$とから成る（$\underline{E} = (\Omega_E, T_E)$）。各属性ａ$\Omega_E$に対して，ｄｏｍ(a)はａの可能な値の集合としての定義域を表わす。Ω_Eの中で，属性＠Ｅは主属性と呼ばれ，ｄｏｍ(＠Ｅ)は，概念データベース内で一意な値（識別子）である。又，$dom(\Omega_E) = dom(@E) \times dom(a_1)$とする。インテグリティ条件$T_E$は，$\Omega_E$が取り得る値の組を定める述語である。Ｅ関係スキーム$\underline{E}$に実際に値を与えたＥ組の集合をＥ関係Ｅ

とする。即ち，

$$E \stackrel{def}{=\!=} \{ e \mid e \in dom(\Omega_E) \wedge T_E(e) \}$$

E組 $e \in E$ の属性 a の値を a(e)とする。属性の並び $a_1, \cdots\cdots, a_m$（$a_i \Omega_E$）の値は，a_1 (e)，…，a_m(e)を並べたもので，$a_1 \cdots\cdots a_m$(e)で表わす。E関係E中で，主属性＠Eの値は一意なので；＠Eは主キーとなる。E関係スキームはT_Eを省略して$\underline{E}$（＠E，a_1，……，a_m）（$m \geq 0$）とも表わす。

R関係スキーム

任意のn（≧2）個のE関係スキーム E_1，…，E_n を考える（この時 E_i は互いに異なっていなくてよい）。この時，ある属性集合 Ω'_R（$\Omega'_R \Omega_E = \emptyset$）があり，ある関係 $dom(\Omega_{E_1}) \times \cdots dom(\Omega_{E_n}) \times dom(\Omega'_R)$ が存在する時に，これをR関係スキーム $\underline{R} = (\underline{E_1}, \cdots, \underline{E}_n, \Omega'_R, T_R)$ で表わす。T_R はインテグリティ条件で，関係に含まれるべき組を定める述語である。$\underline{R}$に実際に値を与えたものをR関係Rとする。

即ち，

$$R \stackrel{def}{=\!=} \{ r \mid r \underset{i=1}{\overset{n}{\times}} dom(\Omega_{E_i}) \times dom(\Omega'_R) \wedge T_R(r) \}$$

各要素 $r \in R$ を，R組とする。

$\Omega'_R = \{a_1, \cdots, a_m\}$（$m \geq 0$）とする。各E関係 E_i において，Ω_{E_i} 内の主属性 $＠E_i$ は主キーであるから，$\underline{R}$内の Ω_{E_i} を $＠E_i$ で表わすことにする。この時，R関係スキームを，T_R を省略して，$\underline{R}$（＠E_1，…，＠E_n，a_1，…，a_m）と書く。又，キー属性は下線を引く。R関係Rは次のようになる。

$$R \stackrel{def}{=\!=} \{ r' \mid r' = ＠E_1 \cdots ＠E_n a_1 \cdots a_m(r) \wedge r \in \underset{i=1}{\overset{n}{\times}} dom(\Omega_{E_i}) \underset{j=1}{\overset{m}{\times}} dom(a_j) \wedge \Gamma_R(r) \}$$

Γ_R の表す条件としては，例えば，常に全てのR組 $r \in R$ に対して，全ての $\underline{E_i}$ のE関係 E_i 内に $＠E_i$（e_i）なるE組 e_i が存在しなければならない。

概念スキーマ$内の各EとR関係の集合を，概念データベースSとする。

概念操作言語（CML）

概念データベースの関係は概念操作言語により操作される。概念操作言語（CML）はQUEL〔HELDG76〕と同等の構文（文法）をもつものである。操作はある目標スキーマを満足する目標集合をデータベースから導出する検索演算と，データベースを変化させる更新演算がある。

検　索

X_1，…，X_lを，S内の1個の任意の関係スキームとし，$\underline{T}(a_1, \cdots, a_k)$を利用者の必要とする目標関係スキームとする。検索とは，Ω_{X_l}，からΩ_Tを定める検索関数$f(\Omega_{X_1} \cdots, \Omega_{X_l}) = \Omega_T$を定義することである。このfを$X = X_1 \times \cdots \times X_l$に作用させる目標関係Rが得られる。

概念操作言語（CML）では，$\mathcal{S}$内の関係X_iの各組は組変数x_iを用いて特定される。組変数は(1)の様に定義される。

range(x_1, X_1)…(x_l, X_l)；　　…… (1)

retrieve into T(tl) where qual …… (2)

検索関数fは，(2)式の様に記述される。Tは目標関係名である。tlは目標リストと呼ばれ，$tl :: = a_1 = exp_1 \cdots, a_k = exp_k$であり，各$exp_i$は属性又は属性上の算術式（e.g. $x_h . a_h + x_j . a_j$）である。$w \in X (= X_1 \times \cdots \times X_l)$に対して，tl(w)は組$(a_1, \cdots, a_m)$を与える関数$(tl : \overset{l}{\underset{i=1}{X}} dom(\Omega_{R_i}) \rightarrow dom(\Omega_T))$と考えられる。qualは，結合述語$jf_{ij} :: = x_i . a_i \ \theta \ x_j . a_j (dom(a_i) = dom(a_j))$と制限述語$rf_i :: = x_i . a_i \ \theta \ v$（vは定数）とから成る論理式である（$\theta \in \{<, \leq, =, \geq, >, \neq\}$）。結合述語は，$jf_{ij} : dom(\Omega_{R_i}) \times dom(\Omega_{R_i}) \rightarrow \{true, false\}$，制限述語は$rf_i : dom(\Omega_{R_i}) \rightarrow \{true, false\}$なる述語である。ここで，$x_k$は関係$X_k$の組変数であり，$x_k . a_k$は$X_k$の属性を表す（k＝i，j）。$w \in X$に対して，qual(w)はtrue又はfalseを与える論理式と考えられる。$(qual : \overset{l}{\underset{i=1}{X}} dom(\Omega_{R_i}) \rightarrow \{true, false\})$即ち，$R = \{ r \mid r = (a_1, \cdots, a_k) = tl(w) \wedge qual(w) \wedge w \in X_1 \times \cdots \times X_l \}$。又，Rは冗長な組を持たない。概念モデルの検索では，主属性が識別子を取る事から，主属性間の比較演算子として＝と≠のみが許され，更に主属性上の算術演算は許されない。*)

基本更新演算

E(@E, a_1, …)とR(@E_1, …, @E_n, b_1, …)を各々，$\mathcal{S}$内の任意のEとR関係とする。この2種の関係に対して，消去，追加，置換があるので，次の6種の基本演算がある。

(1)　del(e, E | e∈E)；組e　Eを除く。同時に，@Eを主属性として持つ全てのR関係Rから@E(e)＝@E(r)なる組rを除く。

(2)　del(r, R | r∈R)；組r∈Rを除く。

(3) app(e,E) ; 関係Eに組eを加える。

(4) app(r, R | $e_1 \in E_1$, …, $e_n \in E_n$) ; 各$E_i(e_i)=@E_i(r)$なる組e_iを持つ(i＝1 , …, n)とき, Rに組rを加える。

(5) rep(e, e′, E | e∈E) ;組e∈Eの非主属性値をe′で置換する。

(6) rep(r, r′, R | r∈R) ;組e∈Rの非主属性値をr′で置換する。

＊) 同様に, 集積関数としては, 〔STONM76〕のcountとanyのみが許される。

2.2.1.2 CODASYLデータベースシステム

ここではCODASYLデータベースシステムについて取り上げる。これはCODASYLモデルのデータ構造が階層モデルのデータ構造を包含し, これが共通モデルである関係モデルで表現可能ならば, 翻えって共通モデルとしての関係モデルの正当性を与えるからである。

CODASYL DBSの階層構造

〔CODAS73〕と〔OLLET78〕に基づいたCODASYL DBSについて考える。CODASYL委員会では, 〔CODAS78〕等の新提案を出している, 実際の商用DBS(例, AIM(ACOM), ADBS (NEC)では, 〔CODAS73〕に近いものがようやく実現されているのが現状である。このため, 〔CODAS73〕について検討することにする。

CODASYL DBSは, 図2.6に示す様な4つの階層から構成されると考えられる。

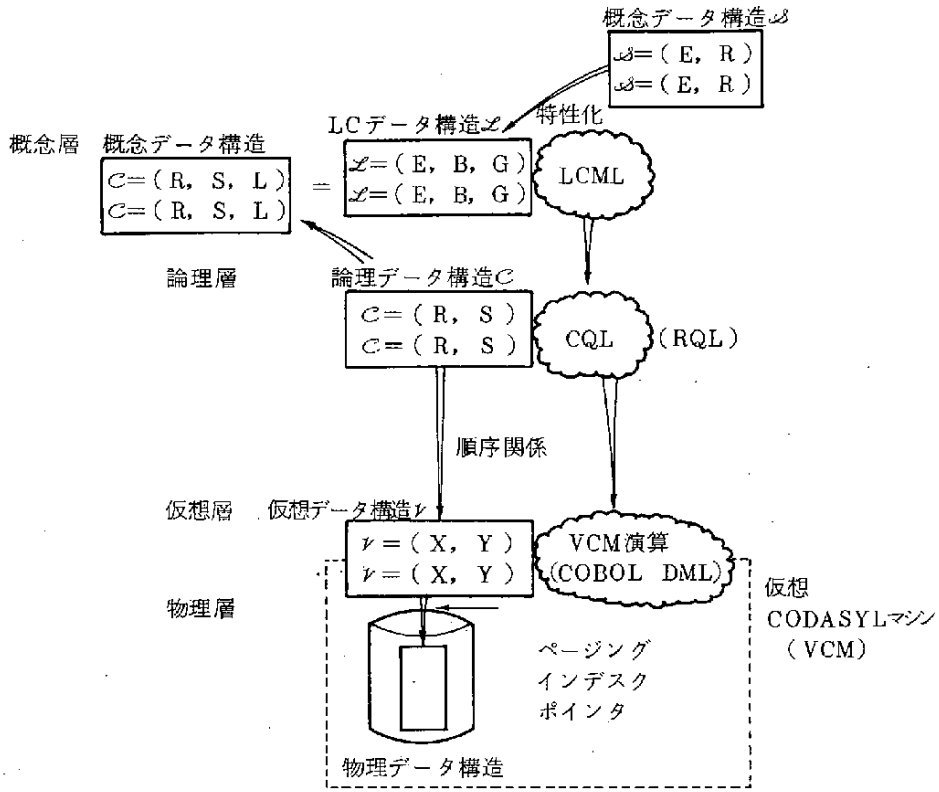


図2.6　CODASYL　DBSの階層化

2.2.1.3　論理CODASYLモデル

ここでは，概念層と仮想層の中間に位置づけられる論理層のデータ構造と，操作演算を定義する。

論理データ構造

CODASYL　DBSの論理層の論理データ構造Cは，時間不変な論

理スキーマ$\underline{C}$と，Cに実際に値を与えた，時間可変な論理データベースCとから成る（C＝（$\underline{C}$，C））。論理スキーマ$\underline{C}$は，レコード型（又はR型）の集合Rと，セット型（又はS型）の集合$\underline{S}$とからなる（$\underline{C}$＝$\underline{R}$，$\underline{S}$））。各レコード型$\underline{R}$は，項目集合Ω_R＝｛@R，t_1，…，t_m｝（$m \geq 0$）と，インテグリティ条件Γ_Rとから成る（$\underline{R}$＝（Ω_R，Γ_R））。各項目$\tau \in \Omega_R$に対して，dom(τ)は，τの取り得る値の集合を表し，τの定義域とする。項目@Rは，特にデータベースキー（dbキー）と呼ばれ，dom(@R)は識別子の集合である。t_iは，データ項目である（i＝1，…，m）。$\pi_R \triangleq$｛t_1，…，t_m｝≦データ項目の集合とする。インテグリティ条件Γ_Rは，Ω_R内の各項目値の組を，意味的に定める述語である。Γ_Rを満足するこの組の集合を，レコード実現値集合（又は，RO集合）Rとする。即ち

$$R \triangleq \{ r \mid r \in dom(\Omega_R) \wedge \Gamma_R(r) \}$$

ここで，$dom(\Omega_R) \triangleq \underset{\forall \tau \in \Omega_R}{X} dom(\tau)$である。

R内の各要素$r \in R$を，レコード実現値（又は，R実現値）とする。各項目$\tau \in \Omega_R$のレコード実現値）とする。各項目$\tau \in \Omega_R$のレコード実現値を，$\tau(r)$と表す。$r \in R$の項目集合｛t_1，…，t_l｝$\leq \Omega_R$の値を，$t_1 \cdots t_l(r)$と表す。全ての$r \in R$に対して，dbキー@Rの値は一意であるので，@Rは$\underline{R}$の主キーとなる。又，任意のレコード実現値（RO）集合RとR′（$\neq$R）において，$\forall r \in R \quad \forall r' \in R' \quad @R(r) \neq @R'(r')$。

よって，@R(r)を$\delta(r)$と表す。又，dbキー値を，利用者は見れないので，@Rを除いてレコード型を$\underline{R}$（t_1，…，t_m）（Γ_Rも省略する）と表す。Rに対するレコード実現値（RO）集合の集合をRとする。

論理データ構造Cでは，レコード実現値（RO）集合間の関係として部分関数を表せる。ある2つのRO集合R_2からR_1にある部分関数fが存在するとき，cでは，これをセット型（又はS型）で表す。セット型$\underline{S}$は，部分関数fの定義域のRO集合のレコード型R_1と値域のレコード型R_2，及びインテグリティ条件Γ_Sとから成る（S＝（$\underline{R_1}$，$\underline{R_2}$，Γ_S））。ここで，$\underline{R_1}$をSの親レコード型，R_2をSの子レコード型と呼び，図2.7の様に$\underline{R_1}$からR_2への有向辺で図示される。インテグリティ条件Γ_Sは，Ω_{R_1}とΩ_{R_2}とから意味的に導出される述語であり，セット型$\underline{S}$として取り得るR_1とR_2

のレコード実現値の対を定める。Sに実際に値を与えたものを，セット実現値集合（又はSO集合）Sとする。

$$S \triangleq \{ s \mid s \in R_1 \times R_2 \wedge \Gamma s(s) \}$$

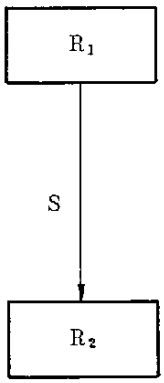


図2.7　セット型S＝（R_1，R_2）

各s＝（r_1，r_2）　Sを，セット実現値（又はS実現値）とし，r_1をr_2の親，r_2をr_1の子レコード実現値とする。セット型Sは，R_2からR_1への部分関数を表すので，各$r_1 \in R_1$のSの子$r_2 \in R_2$は，複数（0以上）存在する。一方，各$r_2 \in R_2$に対するSの親$r_1 \in R_1$は，高々1つ存在するだけである。ここで，次を定義する。

各$r_1 \in R_1$に対して，

$$\underline{member}(r_1) \triangleq \{ r_2 \in R_2 \mid (r_1, r_2) \in S \}$$
$$\triangleq r_1 \text{の子の集合。}$$

各r_2　R_2に対して，

$$\underline{owner}(r_2) \triangleq \{ r_1 \in R_1 \mid (r_1, r_2) \in S \}$$
$$\triangleq r_2 \text{の親} r_1 \text{。}$$

セット型Sにおいて，各$r_1 \in R_1$の子レコード実現値集合member(r_1)内の各レコード実現値r_2のあるデータ項目$t \in \pi_{R_2}$の値が，member(r_1)内で一意であるとき，これはインテグリティ条件Γsの1つとなる。このデータ項目$t \in \pi_{R_2}$を，セット型SのDNA（duplicates are not allowed）項目と呼ぶ。即ち，

$$\exists t \in \pi_{R_2}$$
$$\forall r_1 \in R_1 \ \forall r_2, r_2' \in member(r_1)(r_2 \neq r_2')$$
$$t(r_2) \neq t(r_2')$$

レコード型$\underline{R_1}$とR_2において，各々dbキー@R_1と@R_2は主キーとなるので，セット実現値s＝(r_1, r_2) Sを，レコード実現値r_1とr_2のdbキー値の対($\delta(r_1)$, $\delta(r_2)$)としても表せる。よって，セット実限値集合Sを，次の様にする。

$$S \triangleq \{ s' \mid s' = (\delta(r_1), \delta(r_2)) \wedge s = (r_1, r_2) \ R_1 \times R_2 \wedge \Gamma s(s) \}$$

Sに対するセット実現値（SO）集合の集合をSとする。

論理スキーマ$\underline{C}$＝（$\underline{R}$, $\underline{S}$）に対する論理データベースCはRO集合の集合Rと，SO集合の集合Sとから成る（C＝（R, S））。

論理スキーマ内の各セット型$\underline{S}$＝（$\underline{R_1}$, $\underline{R_2}$, Γs）に，論理データベースを変化させる更新演算に対する制約としての型（色）を定義出来る。検索利用だけの場合は，型は不要である。stype(S)は，セット型$\underline{S}$の型を表すとする。型としては，M(andatory)／A(utomatic)とO(ptional)／M(anual)との2種がある。*)

$$stype(S) \quad \{M/A, O/M\}$$

型づけされたセット型から成る論理スキーマ$\underline{C}$は，次の制約を持つ。

〔スキーマ制約〕

スキーマ$\underline{C}$は，全てのセット型$\underline{S}$の型がM／A（stype($\underline{S}$)=M／A）なる有向閉路を含んではならない。□

---

*) S型$\underline{S}$のメンバシップクラスは〔CODAS73〕では消去演算に対する除去クラスrc($\underline{S}$)と，追加演算に対する格納クラスsc($\underline{S}$)とがある。rc($\underline{S}$)∈{M(andatory), O(ptional)} sc($\underline{S}$)∈{A(utomatic), M(anual)}であり，mc($\underline{S}$)＝rs($\underline{S}$)＝rs($\underline{S}$)／sc($\underline{S}$) {M／A, M／M, O／M, O／A}である。
しかし，一般にはmc($\underline{S}$)∈{M／A, O／M}だけが用いられる。

論理操作演算一検索

前に定義した論理データ構造上の論理操作演算を定義する。まず，検索について考える。$\underline{R}_1$，…，$\underline{R}_h$を論理スキーマ内の任意のh（≧1）個のレコード型とする（互いに異っている必要はない）。$\underline{R}=\{\underline{R}_1, \cdots, R_h\}$とする。$\underline{S}_1$，…，$\underline{S}_P$（ここで，$S_i=(R_{i_1}, R_{i_2}, \Gamma s_i)$，$R_{i_1}$，$R_{i_2} \in R$）を，$\underline{C}$内の任意のp個セット型（互いに異っている必要はない）とする。利用者の必要とする目標集合Tは，属性集合$\Omega_T=\{a_1, \cdots, a_k\}$（k≧1）から成るとする。論理検索演算とはf（$\Omega_{R_1}$，…，$\Omega_{R_2}$，$\underline{S}_1$，…，$\underline{S}_P$）$=\Omega_T$なる検索関数fを定義することである。fを直積$R_1 \times \cdots \times R_l \times S_1 \times \cdots \times S_P$に作用させると，$T \subseteq dom(\Omega_T)$が得られる。

論理操作演算－更新演算

論理データベースC上の更新演算について考える。まず，論理スキーマ$\underline{C}$内の各セット型は，型づけがなされているとする。この時，任意のレコード実現値（RO）集合をRとする。R内の各レコード実現値r（∈R）に対して，r及び，rのM／A型のセット型$\underline{S}=(\underline{R}, \underline{R}', \Gamma s)$の全ての子$r' \in R'$，更に，r'のM／A型セット型S'の子$r'' \in R''$，……の集合をmr(r)とする。各$r' \in mr(r)$を含む全てのセット実現値の集合をss(r')とし，$\forall r' \overset{\cup}{mr(r')}$ ss(r')＝sr(r)とする。
この時，

$$ms(r) \triangleq mr(r) \cup sr(r)。$$

又，論理スキーマ$\underline{C}$内の各レコード型$\underline{R}$に対して，$\underline{R}$を子とする全てM／A型セット型$\underline{S}=(\underline{R}', \underline{R}, \Gamma s)$の（$\underline{S}$，$\underline{R}'$）の対の集合を$O_S(\underline{R})$とする。

基本更新演算

$\underline{R}$，$\underline{S}=(\underline{R}_1, \underline{R}_2, \Gamma s)$を，論理スキーマ$\underline{C}$内の任意のレコード型とセット型とする。このとき，レコード実現値（RO）集合Rと，セット実現値（SO）集合Sとに対して，各々，実現値の消去，追加，置換の更新基本演算がある。以下に更新基本演算の意味を示す。

(1) del(r,R)；Rからrを除くとともに，ms(r)を論理データベースCから除く。

(2) del(s,S)；Rからrを除く。stype($\underline{S}$)＝M／Aのとき，$s=(\delta(r_1),\ \delta(r_2))$ Sとすると，$ms(r_2)$もCから除かれる。

(3) app′(r, R | $r_1 \in R_1, \cdots, r_n \in R_n$)；$Os(\underline{R})=\{(\underline{S}_1, \underline{R}_1), \cdots, (\underline{S}_n, \underline{R}_n)\}$とする。rをRに加える為には，まず，各$R_i$内にある$r_i$が存在せねばならない(i=1, …, n)。存在するならば，rをRに加えると共に，各S_iに$(\delta(r_i), \delta(r))$を追加する。rのdbキー$\delta(r)$はDBSによって自動的にセットされるとする。

(4) app(s=$(\delta(r_1), \delta(r_2))$, S | $r_1 \in R_1, r_2 \in R_2$；
$\underline{S}$はO／M型(stype($\underline{S}$)＝O／M)でなければならない。追加しようとする実現値$s=(\delta(r_1), \delta(r_2))$は，$r_1 \in R_1$でかつ$r_2 \in R_2$ならばSに追加される。

(5) rep(r, r′, R)；R内のレコード実現値rのデータ項目値を，rに置換する。

(6) rep(s=$(\delta(r_1), \delta(r_2))$, s′=$(\delta(r_1'), \delta(r_2))$, S | $r_1' \in R_1$)；
S内のセット実現値$s=(\delta(r_1), \delta(r_2))(\in S)$の親を，ある$r_1' \in R_1$にかえる。

2.2.1.4 仮想CODASYLモデル

ここでは，論理層と物理層の中間にある仮想層について考える。仮想層の仮想データ構造を定義する。

仮想データ構造

2.2.1.3で定義した論理データ構造C＝($\underline{C}$, C)の仮想層の表現を仮想データ構造V＝($\underline{V}$, V)とする。$\underline{V}$を論理スキーマ$\underline{C}$に対する仮想スキーマ，Vを論理データベースCに対する仮想データベースとする。Vは，Cに対して，順序関係が導入されている。

まず，C内の各レコード型R＝（Ω_R，Γ_R）に対して，V内の仮想レコード型（又は，VR型）X＝（Ω_X，Γ_X，$<_X$）が定義される。ここで，Xの項目集合Ω_Xとインテグリティ条件Γ_Xとは，各々Ω_XとΓ_Xと同じである。（$\Omega_X=\Omega_R$，$\Gamma_X=\Gamma_R$）。レコード実現値（RO）集合R＝{r | r∈dom(Ω_R)∧Γ_R(r)}を，順序関係$<_X$によって全順序づけたものを，仮想レコード実現値（VRO）集合Xとする。各レコード実現値r∈R と1対1対応するx∈Xを，rの仮想レコード実現値（又は，VR実現値）とする。

このとき，$\forall\tau\in\Omega_R$（$=\Omega_X$）に対して，τ(r)＝τ(x)である。X内のVR実現値は，そのdbキーδによって全順序づけられる。即ち，

$$\forall x,\ x'\in X\ (x\neq x')\ x<_X x'\ \underline{\text{iff}}\ \delta(x)<\delta(x')$$

仮想データ構造Vでは，Cとは独立に，dbキーだけから成り，1つのVR実現値から成るVRO集合X_0（スキーム$\underline{X_0}$＝（Ω_{X_0}），Ω_{X_0}＝{@X_0}）を定義出来る。これを，システムVRO集合と呼ぶ。$\underline{X_0}$をシステムVS型とする。

論理スキーマC内の各セット型S＝（$\underline{R_1}$，$\underline{R_2}$，Γ_S）に対して，仮想スキーマ内に仮想セット型（又は，VS型）Y＝（$\underline{X_1}$，$\underline{X_2}$，Γ_Y，$<_Y$）を定義する。ここで，$\underline{X_1}$と$\underline{X_2}$は，各々，レコード型$\underline{R_1}$と$\underline{R_2}$に対応する仮想レコード型である。Γ_Yは，Yのインテグリティ条件で，Γ_Sと等しい（$\Gamma_Y=\Gamma_S$）。

$\underline{X_1}$をYの親，$\underline{X_2}$をYの子という。ここで，各$x_1\in X_1$に対して，以下を定義する。

$m_Y(x_1)$≜{x_2 | $x_2\in X_2\wedge$（x_1とx_2は各々，$r_1\in R_1$と$r_2\in R_2$のVR実現値）∧（r_1，r_2）∈S}

$m_Y(x_1)$を，$x_1\in X_1$の子仮想レコード（VR）実現値の集合とする。

各$x_2\in m_Y(x_1)$に対する$x_1\in X_1$を，x_2の親とする。この時，Sのセット実現値（SO）集合Sに対応した仮想セット実現値（VSO）集合Yが次の様に定義される。

$$Y \triangleq \{ (x_1, m_Y(x_1)) \mid x_1 \in X_1 \ \wedge \ m_Y(x_1) \neq \phi \}$$

各 $y = (x_1, m_Y(x_1)) \in Y$ を，仮想セット（ＶＳ）実現値とする。集合 $m_Y(x_1)$ は，全順序づけられている。順序関係 $<_Y$ としては，次の3種がある。

(i) あるデータ項目 $t \in \pi x_2$ に対して，$m_Y(x_1)$ 内のＶＲ実現値を昇順に順序づける。

(ii) (i)に対して，降順に順序づける。

(iii) ＤＢＳが自動的に順序づける。

(i)又は(ii)におけるデータ項目 t を，ＶＳ型 $\underline{Y}$ の整列データ項目とする。

任意の仮想レコード（ＶＲ）型 $\underline{X} = (\Omega_X, \Gamma_X, <_X)$ の仮想レコード実現値（ＶＲＯ）集合Xが，あるデータ項目 $t \in \pi_X$ による全順序関係を持つとき，$\underline{X}_0$ と $\underline{X}$ とに仮想セット（ＶＳ）型 $\underline{Y} = (\underline{X}_0, \underline{X}, \Gamma_Y, <_Y)$ を定義できる。この仮想セット実現値（VSO）集合Yは，$Y \triangleq \{ (x_0, X) \}$ である。この様なＶＳ型は，単項ＶＳ型と呼ばれる。Yを単項ＶＳＯ集合とする。

ここで，各ＶＳ型 $\underline{Y} = (\underline{X}_1, \underline{X}_2)$ に対して，次の述語 $P_Y : dom(\Omega_{X_2}) \rightarrow \{ true, false \}$ を定義する。

$$P_Y(x_1, x_2) = \begin{cases} true \ \underline{if} \ x_1 \in X_1 \wedge x_2 \in X_2 \wedge x_2 \in m_Y(x_1) \\ false \ \underline{otherwise} \end{cases}$$

仮想レコード実現値（ＶＲＯ）集合と，仮想セット実現値（ＶＳＯ）集合との集合を，仮想スキーマ $\underline{V}$ の仮想データベースVとする。

以上より，論理データ構造Ｖとの間には，以下の性質が存在することは明らかである。

〔性質１〕

仮想データ構造Vから，順序関係，システムＶＲ型とＶＲＯ集合，及び単項ＶＳ型とＶＳＯ集合を除いたものは，論理データ構造Ｃである。□

仮想操作演算

仮想CODASYL データ構造の操作を与えるため，データ操作言語（D

ＭＬ）〔ＯＬＬＥＴ７８〕を直接実行出来る仮想的なＣＯＤＡＳＹＬ マシンを考える。これは，多テープチューリングマシンで表わされ，全てのＣＯＢＯＬＤＭＬはこの上でシュミレートできる〔ＴＡＫＩＭ８２ｂ〕。これは，５章のＬＩＰのＶＯＰ演算と同じものである。（ただし，ここでは更新演算も含まれている。）各操作については，表２.１を参照のこと。

2.2.1.5　概念ＣＯＤＡＳＹＬモデル

ＣＯＤＡＳＹＬ　ＤＢＳの最上位層である概念層について論じる。概念層では，論理データ構造を制約した概念データ構造と，概念データ構造上の基本操作演算について論じる。

概念データ構造

論理データ構造Ｃは，レコード型と，レコード型間の部分関数を表すセット型とから成る論理スキーマ$\underline{C}$と，レコード型とセット型の各々に，実際に値を与えたレコード実現値（ＲＯ）集合とセット実現値（ＳＯ）集合とから成る論理データベースＣとの対である（Ｃ＝（$\underline{C}$，Ｃ）。論理スキーマ$\underline{C}$内のあるｎ（≥ 2）個のレコード型$\underline{R_1}$，…，$\underline{R_n}$（互いに異っている必要はない）が，共通の子レコード型$\underline{R}$を持つｎ個のセット型$\underline{S_i}$＝($\underline{R_i}$，$\underline{R}$，Γ_{S_i}）の親であり，かつ部分関数ｆ：dom（Ω_{R_1}）×…×dom（Ω_{R_n}）→dom（Ω_R）が存在しているとする。このとき，概念データ構造では，論理データ構造の２つのレコード型間の部分関数に加えて，このｎ個のレコード型間の関係も表される。概念層では，このセット型$\underline{S_i}$（$i=1$，…，n）と$\underline{R}$とをまとめてリンク型Ｌ＝（$\underline{R_1}$，$\underline{S_1}$（$\underline{R_1}$，$\underline{R}$），……，$\underline{R_n}$，$\underline{S_n}$（$\underline{R_n}$，$\underline{R}$），$\underline{R}$，Γ_L）とする。〔図2.8〕。Γ_Lは，Ω_{R_1}，…，Ω_{R_n}，Ω_R，から意味的に導出されるインテグリティ条件である。リング型$\underline{L}$に，実際に値を与えたものを，リンク実現値（ＬＯ）集合Ｌとする。

表 2.1　仮想操作演算

VCM演算	COBOL DML
first (X);	find first X within A.
last (X);	find last X within A.
next (X);	find next X within A.
prior (X);	find prior X within A.
CALC(X);	find any X.
first (Y);	find first X_2 within Y.
last (Y);	find last X_2 within Y.
next (Y);	find next X_2 within Y.
prior (Y);	find prior X_2 within Y.
owner (Y);	find owner within Y.
get (X);	get X.
if SR=n, go to L;	if SR=n go to L.
open (T_i);	open T.
close (T_i);	close T.
write (T_i);	write T.
stop;	stop.
accept (X, β_v);	accept β_v from X current.
accept (Y, β_v);	accept β_v from Y current.
find (X, β_v);	find X; db-key is β_v.
store (X);	store X.
erase (X);	erase X.
modify (X);	modify X including Y_1, …, Y_n
connect (Y);	connect X_2 to Y.
disconnect (Y);	disconnect X_2 from Y.
modify (Y);	modify X_2 only Y.
P (β_1, …, β_n)	P (β_1, …β_n)
go to L;	go to L.
$\beta_i \leftarrow \beta_j$;	move β_j to β_i

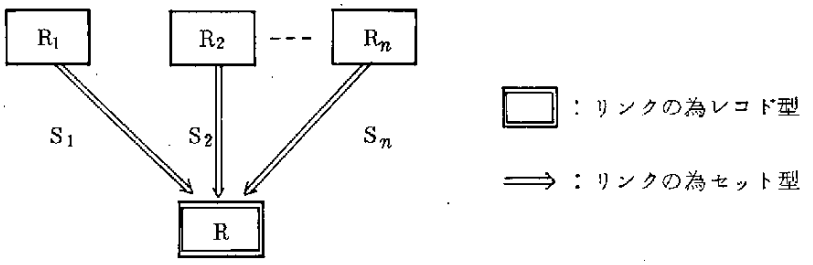


図2.8　リンク型$\underline{L}=\{\underline{R}, \underline{S_1}(\underline{R_1}, \underline{R}), \cdots, \underline{S_n}(\underline{R_n}, \underline{R})\}$

$$L \triangleq \{ l \mid l=(r_1, \cdots, r_n, r) \in R_1 \times \cdots \times R_n \times R \wedge \bigwedge_{i=1}^{n} (r_i, r) \in S_i \wedge \Gamma_L(r_1, \cdots, r_n, r) \}$$

各$l \in L$をリンク（L）実現値とする。$\underline{L}$を表すセット型$\underline{S_i}$をリンクの為のセット型，各S_iの共通の子レコード型$\underline{R}$をリンクの為のレコード型とする。よって，概念スキーマ$\underline{D}$は，レコード型$\underline{R}$，セット型$\underline{S}$，及びリンク型$\underline{L}$の3種の型から成る。概念データベースDも$\underline{R}$のレコード実現値（RO）集合，$\underline{S}$のセット実現値（SO）集合，$\underline{L}$のリンク実現値（LO）集合の3種の実現値集合から成る。

ここで，全てのリンクの為のセット型の型は，M/A(stype($\underline{S}$)=M/A)であるとする。＊）

〔命題2.1〕

全ての概念CODASYLデータ構造は，論理CODASYLデータ構造で表せる。

〔証　明〕

概念CODASYLデータ構造のリンク型$\underline{L}$は，セット型とレコード型で表せるので明らかである。

＊）〔CODAS73〕ではR型$\underline{R_1}$とR_2間に複数のS型$\underline{S_1}$（$\underline{R_1}$，$\underline{R_2}$）（i＝1，…，n，n≧2）あるとき，mc（S_i）＝M／A　であれるのは高々1つとなる。そうでないと，各S型S_iは皆同一の親子関係をもつことになってしまう。

基本操作演算

データベースD内の任意のRO集合をRとする。各r∈Rに対して，その全てのM／A型S実現値の子r′∈R′の集合が定まる。同様にr′∈R′のM／A型S実現値の子r″∈R″の集合が定まる。rから始めて，この様に再帰的に定義されていく全てのR実現値の集合をM(r, R)とする。又M(r, R)内の各r′∈R′を親又は子とする全てのS実現値の集合をO(r′)とする。この時，ms(r, R)≦(r, R)U $\underset{\forall r' \in M(r, R)}{}$ O(r′)。又，Dの各レコード型Rに対して，Rを子とする全てのM／Aセット型S′とその親R′の対の集合をO_S(R)とする。R, S(R₁, R₂), {L, S_i(R_i, L)(i=1, …, n)}を各々，D内の任意のRO，SO，LO集合とすると，次の9種の論理演算がC上に存在する。

(1) del(r, R | r∈R)；ms(r, R)をDから消去する。

(2) del(s, S | s∈S)；s=(k_1, k_2)∈Sの時，mc(S)=M／Aならば，ms(r_2, R_2)(dbk(r_2)=k_2)を，O／Mならばsのみを消去する。

(3) del(l, L | l∈L)；l∈L及び，各S_iからlを子とするS実現値S_iを消去する。

(4) app(r, R | r_1∈R_1, …, r_n∈R_n)；全ての(S_i, R_i)∈O_S(R)に対して，あるr_i∈R_iの時，rをRに加えると共に，各S_iにS実現値($\delta(r_1)$, $\delta(r)$)を加える。

(5) app(s, S | r_1∈R_1, r_2∈R_2)；mc(S)=O／Mの時，r_1∈R_1, r_2∈R_2ならば，Sに($\delta(r_1)$, $\delta(r_2)$)を追加する。

(6) app(l, L | r_1∈R_1, …, r_n∈R_n)；RをLとした(4)と同じ。

(7) rep(r, r′, R | r∈R)；r∈Rのデータ項目値をr′に置換する。

(8) rep(s, s′, S | s∈S)；s=(k_1, k_2)∈Sをs′=(k_1', k_2)($k_1' \neq k_1$)に置換する。即ち，子に対する親を置換する。

(9) rep(l, l′, L | l∈L)；l∈Lのデータ項目値をl′に置換する。

2.2.1.6　ローカル概念(CODASYL)モデル

概念モデルのR関係を型づけ，型毎に更新演算の意味(範囲，条件)を与える事によって，CODASYLモデルと等価なモデル(ローカル概念

（ＬＣ）モデル）が構成出来る事を以下に示す。

ＬＣモデルのデータ構造

ローカル概念（ＬＣ）データ構造

概念データ構造のＲ関係スキームを以下の様に特性化したものをローカル概念（ＬＣ）データ構造と呼ぶ。

(1) Ｒ関係スキーム$\underline{R}$として，ＢとＧ型の２種を設ける。$\underline{R}$がＥ関係E_2からE_1への部分関数を表す時，Ｂ型とする。@E_2の値によってＲ内の組を一意に識別出来るので，@E_2は主キーとなる。

これを$\underline{R}$(@E_1, @$\underline{E_2}$)と記す。$\underline{R}=(\underline{E_1}, \cdots, \underline{E_n}, \Omega_{R'}, \Gamma_R)$が部分関数，$\overset{n}{\underset{i=1}{X}}\ dom(\Omega_{E_i}) \rightarrow dom(\Omega_{R'})$を表す時，$\underline{R}$をＧ型とする。

この時，Ｒ内の各組は主属性集合｛@E_1, …, @E_n｝値によって識別できるので，主属性集合が主キーとなる。これを$\underline{R}$(@$\underline{E_1}$, …, @E_n, a_1, …, a_m)と記す。

(2) 更に，Ｂ型の$\underline{R}$には，Ｔ（全）とＰ（部分）の２つ型があり，更新に対して異なった制約を与える。

以上の特性を持つスキーマをローカル概念スキーマ（ＬＣＳ）$\underline{\mathcal{L}}$，これに実際の値を与えたものをローカル概念（ＬＣ）データベース とする。□

ＬＣモデルと概念ＣＯＤＡＳＹＬモデルのデータ構造は，次の様に対応づけられる〔表2.2〕

〔ＬＣデータ構造と概念ＣＯＤＡＳＹＬデータ構造の対応規則〕

(1) レコード型Ｒ(t_1, …, t_m)とＥ関係スキーム$\underline{R}$(@Ｒ, t_1, …, t_m)が，データ項目t_iと属性t_i，dbキーと主属性@Ｒの様に対応する。任意のＥ関係スキーム$\underline{R}$(@$\underline{R}$, …)と$\underline{R'}$(@$\underline{R'}$, …)(Ｒ≠Ｒ′)に対して，dom(@Ｒ)≠dom(@Ｒ′)である。

(2) セット型$\underline{S}$($\underline{R_1}$, $\underline{R_2}$)とＢ型Ｒ関係スキームＳ(@R_1, @R_2)が対応する。ここで，@R_iはレコード型$\underline{R_i}$に対応したＥ関係スキームの主キーである(ｉ＝1,2)。ＳＯ集合ＳはR_2からR_1への部分関数であるので，この関係をそのままＲ関係に対応づける。この時部分関数の値域を表す@R_2が主キーとなる。スキームは，セット型がＭ／Ａの時Ｔ型に，Ｏ／Ｍの時Ｐ型になる。

(3) リンク型$\underline{L}=\{\underline{R}(t_1, \cdots, t_m), \underline{R_i}, \underline{S_i}(R_i, \underline{R})(i=1, \cdots, n)\}$（$n \geq 2$, $m \geq 0$）とG型R関係スキーム$\underline{L}(@\underline{R_1}, \cdots, @\underline{R_n}, t_1, \cdots, t_m)$が対応する。*) $@R_i$はレコード型$\underline{R_i}$に対応したE関係スキームの主キーである（$i=1, \cdots, n$）。LO集合$L=\{L, S_1, \cdots, S_n\}$は，RからR₁×…×Rₙへの単射であり，主属性集合$\{@R_1, \cdots, @R_n\}$によってL×R₁×…×Rₙの各組を識別出来るので，これが主キーとなる。□

以上の対応規則より，次の命題が成り立つ事は明らかである。

〔命題2.1〕

任意の概念CODASYLデータ構造と，(1)~(3)によって導かれるLCデータ構造とは1対1対応し，データ構造は等価である。■

表2.2　LCと概念CODASYLデータ構造の対応

概念CODASYLデータ構造	LCデータ構造
レコード $\underline{R}(t_1, \cdots, t_m)$	E関係スキーム $\underline{R}(@\underline{R}, t_1, \cdots, t_m)$
M／Aセット型 $R_1(t_{11}, \cdots, t_{1m_1})$ S（M／A） $R_2(t_{21}, \cdots, t_{2m_2})$	E関係スキーム $\underline{R_1}(@\underline{R_1}, t_{11}, \cdots, t_{1m_1})$ $\underline{R_2}(@\underline{R_2}, t_{21}, \cdots, t_{2m_2})$ T関係スキーム $\underline{S}(@\underline{R_1}, @\underline{R_2})$
O／Mセット型 $R_1(t_{11}, \cdots, t_{1m_1})$ S（O／M） $R_2(t_{21}, \cdots, t_{2m_2})$	E関係スキーム $\underline{R_1}(@\underline{R_1}, t_{11}, \cdots, t_{1m_1})$ $\underline{R_2}(@\underline{R_2}, t_{21}, \cdots, t_{1m_1})$ P関係スキーム $\underline{S}(@\underline{R_1}, @\underline{R_2})$
リンク型 $R_1(t_{11}, \cdots, t_{1m_1})$ … $R_n(t_{n1}, \cdots, t_{nm_n})$ S_1　S_n $R(t_1, \cdots, t_m)$	E関係スキーム $\underline{R_i}(@\underline{R_i}, t_{i1}, \cdots, t_{im_i})$ $(i=1, \cdots, n)$ G関係スキーム $\underline{L}([\underline{S_1}]@\underline{R_1}, \cdots, [\underline{S_n}]@\underline{R_n}, t_1, \cdots, t_m)$

*) $@R_i=@R_j$（$i \neq j$）の時には，$S_i @R_j$，$S_j @R_j$を属性名とする。

ＬＣモデルの操作演算

本節では，ローカル概念（ＬＣ）モデルの操作演算について考える。まず，ＬＣデータ構造上の基本更新演算を定義する。

ＬＣモデルの基本演算

ＬＣデータベース 内の任意のＥ関係をＲ（スキームを$\underline{R}=(\Omega_R=\{@\underline{R}, t_1, \cdots\})$）とし，Ｅ関係Ｒ内の各組をｒとする。命題2.1より，ＬＣデータ構造とＣＯＤＡＳＹＬデータ構造は1対1対応するので，集合$t_S(r, R)$と$T_S(\underline{R})$を，各々2.2.1.3のms（ｒ，Ｒ），$O_S(\underline{R})$と同一に定義出来る。Ｒ（@Ｒ，$t_1, \cdots$），Ｓ（$@R_1$，R_2），Ｌ（$@R_1, \cdots, @R_n, b_1\cdots$）を各々，内の任意のＦ，Ｂ，Ｇ型関係とする。データ構造が特性化されたので，ＣＭＬ〔2.2.1.1〕に対して次の9種の基本演算が存在する。

〔ＬＣ基本更新演算〕

(1) del（ｒ，Ｒ｜ｒ∈Ｒ）；からt_S（ｒ，Ｒ）を消去する。

(2) del（ｓ，Ｓ｜ｓ∈Ｓ）；ＳがＴ型の時，$@R_2(s)=@R_2(r_2)$なる$t_S(r_2, R_2)$を消去する。Ｐ型の時，ｓのみを消去する。

(3) del（1，Ｌ｜1∈Ｌ）；組1∈Ｌを消去する。

(4) app（ｒ，Ｒ｜$r_1\in R_1, \cdots, r_n\in R_n$）；全ての$(\underline{S_i}, \underline{R_i})\in T_S(\underline{R})$に対して，ある組$r_i\in R_i$の時，Ｒにｒを加えると共に$@R_i(r_i)=@R_i(s_i)\wedge @R(s_i)=@R(r)$なる$s_i$を$S_i$に加える（ｉ＝1，…，ｎ）。

(5) app（ｓ，Ｓ｜$r_1\in R_1$，$r_2\in R_2$）；ＳがＰ型で，$r_1\in R_1$，$r_2\in R_2$の時$@R_1(r_1)=@R_1(s)\wedge @R_2(s)=@R_2(r_2)$なる組ｓをＳに加える。

(6) app（1，Ｌ｜$r_1\in R_1, \cdots, r_n\in R_n$）；全ての$R_i$に対して，ある$r_i\in R_i$の時，$@R_i(r_i)=@R_i(1)$（ｉ=1，…，ｎ）なる組1をＬに加える。

(7) rep（ｒ，ｒ′，Ｒ｜ｒ∈Ｒ）；(8) rep（ｓ，Ｓ′，Ｓ｜ｓ∈Ｓ）；(9) rep（1，1′，Ｌ｜1∈Ｌ）；各々，ｒ∈Ｒ，ｓ∈Ｓ，1∈Ｌの非主キー値をｒ′，ｓ′，1′で置換する。□

〔命題2.2〕

ＬＣモデルの(1)～(9)と，概念ＣＯＤＡＳＹＬモデルの(1)～(9)の基本演算

は，この順で１対１対応する。

〔証明〕

命題2.1より，集合ms（r，R）とts（r，R），Os（$\underline{R}$）とTs（$\underline{R}$）とは互いに１対１対応する。よって，ＬＣの(i)とＣＯＤＡＳＹＬの(i)の基本演算とは，等価な更新条件と更新効果を持つ事（ i＝1，…；９） は，明らかである。□

命題2.1と命題2.2より，次の定理が成り立つ。

〔定理2.1〕

ＬＣモデルと概念ＣＯＤＡＳＹＬモデルとは１対１対応し，等価である。

2.2.2 分散性問題

分散型データベースシステム（ＤＤＢＳ）が，複数のデータベースシステム（ＤＢＳ）を通信網を用いて相互結合して構成されることから生じる分散性問題としては次の２点がある。

(i) 設計問題

(ii) 分散検索／更新問合せ処理

(i)の問題としては，ローカル概念（ＬＣ）データ構造と，全体概念（ＧＣ）データ構造との間の写像をどのように行うかである。統合的設計方法では，各ＤＢＳ上に，ＤＤＢＳ全体で共通なデータモデルに基づいたＬＣデータ構造がまず定義されていて，これらのＬＣデータ構造上に，ＧＣデータ構造が定義される。逆に分割型設計方法では，まずＧＣデータ構造が定義されて，その後に全体の利用者の利用要求に会う様に，ＧＣデータ構造を各データモヂュール（ＤＭ）に分割してＬＣデータ構造とされる。我々のＪＤＤＢＳでは，前者の統合的設計方法を用いている。

(ii)では，ＧＣデータ構造に基づいた演算（全体概念（ＧＣ）演算）を，どの様にＬＣデータ構造上の演算と，各データモヂュール（ＤＭ）間の通信処理とによって表すかが問題となる。ＧＣ演算としては，検索と更新演算とがある。検索演算の分散処理では，各ＤＭ内のローカル処理とＤＭ間での通信処理コストをいかに最少化していくかが問題になる。通信網の速度，信頼性，通信形態（１対１通信又は１対多（放送）通信）が，分散検索処理を考える上で重要である，我々のＪＤＤＢＳでは，放送通信に基づ

いた検索の処理方式を用いている。更新演算の分散処理では，各トランザクションの複数のＤＭへの更新演算の原子性を保つ為のコミットメント制御と（Comitment control），各ＤＭへの複数のトランザクションからの検索と更新演算の実行順序をなるべく並行度を高めながらかつＤＤＢＳ全体のインテグリティを保つ為の同時実行制御（concurrency control）とが必要である。

更に，セキュリティ，障害回復の制御機能も必要となる。ＪＤＤＢＳではこれらの問題のうち，コミットメント制御方法について，放送通信と各ＤＭの分散制御に基づいた方法を用いている。

2.2.2.1　全体概念データ構造

統合化とは，各データモヂュール（ＤＭ）内のローカル概念（ＬＣ）データ構造上に，DDBSのある共同体にとって必要な一意な全体概念(GC)データ構造を定義し，ＧＣデータ構造とＬＣデータ構造との対応情報を分散情報（ＤＩ）として生成する過程である〔図2.9〕

ＧＣデータ構造の定義は，ＤＤＢＳの全体管理者（ＧＡ）が行う。

ＧＣデータ構造

全体概念（ＧＣ）データ構造Ｇは，ＧＣ関係スキームの集合としてのＧＣスキーマ（ＧＣＳ）と，Ｇ内の各ＧＣ関係スキームに実際に値を与えたＧＣ関係の集合としてのＧＣデータベースＧとから成る。Ｇ内の各ＧＣ関係スキームＲは，属性集合$\Omega_R=\{a_1,\cdots,a_m\}$（$m\geq 1$）とインテグリティ条件$\Gamma_R$とから成る。（$R=(\Omega_R,\Gamma_R)$）各属性$a_i\in\Omega_R$の取り得る値の集合を定義域とし，$dom(a_i)$と表す。$\Gamma_R$を省略して，ＧＣ関係スキームを$R(a_1〔/dom(a_1)〕,\cdots,a_m〔/dom(a_m)〕)$と表すことにする。

各Ｒに実際に値を与えたものをＧＣ関係$R=\{r\mid r\in dom(a_1)\times\cdots\times dom(a_m)\wedge\Gamma_R(r)\}$とする。各要素$r\in R$を，組（又はＧＣ組）と呼び，rの属性$a_i\in\Omega_R$の値を$a_i(r)$と表す。

ＧＣデータ構造は，通常の関係データ構造である。

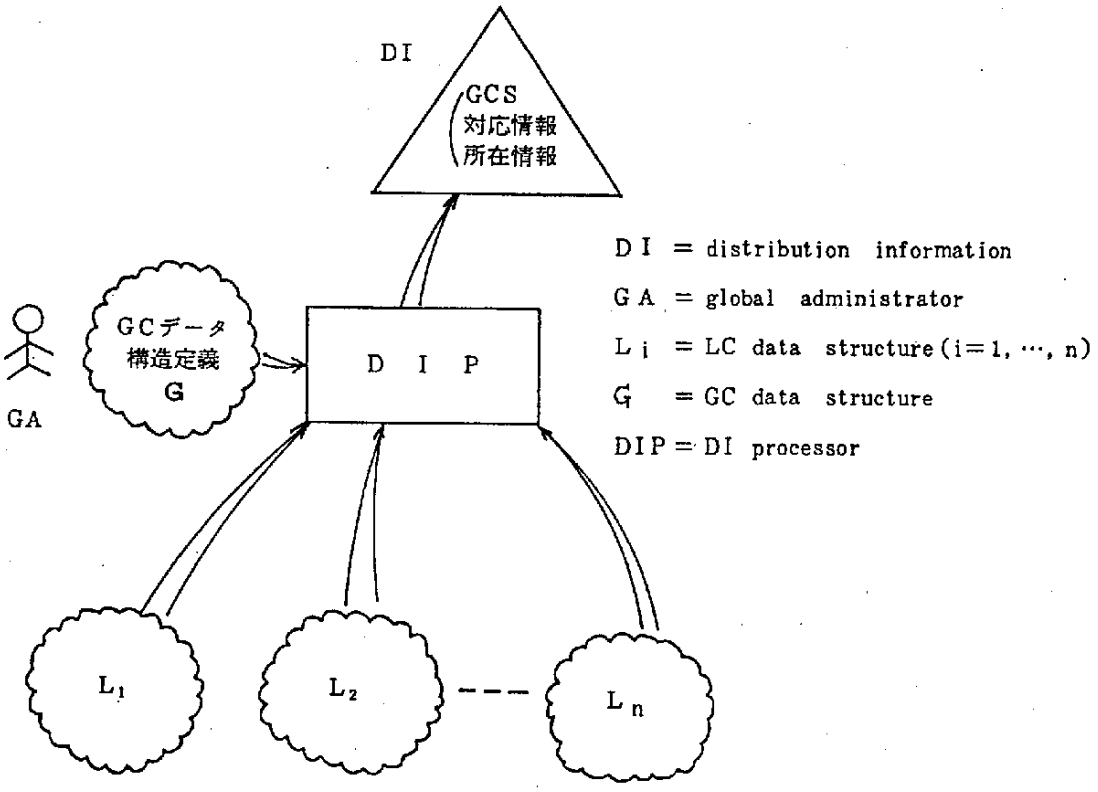


図 2.9　統合化

GC演算

GCデータベースに対する検索演算を，ローカル概念（LC）層と同じ構文の関係述語言語RQLによって表す。RQLは，QUELと同様の構文を持ち，次の様に記述される。

range　（g_1, G_1）……（g_l, G_l）；

retrieve into T (t1) where qual ;

ここで，G_iは全体概念（GC）スキーマ内のGC関係スキーム名である（i=1, …, l）。

range文は，問合せの参照する各G_iに対して，GCデータベース内のGC関係G_iの組を取る組変数g_iを定義する。問合せ内でG_iの属性aは，

gi．aと表される。

Tは，利用者の必要とする目標関係スキーム（$I=(\Omega_T)$，$\Omega_T=\{a_1, \cdots, a_k\}$）名である。

tlは目標リストと呼ばれ，次の様に書かれる。

$$tl ::= a_1 = exp_1, \cdots, a_k = exp_k$$

exp_iは，定数，属性，又はこれらの上の算術式である。t_lは，$G_1 \times \cdots \times G_l$ からTへの関数と考えられる。ここで，$G_i = dom(\Omega_{G_i})$，$T = dom(\Omega_T)$である。

qualは，条件式（qualification）と呼ばれ，次の様に積正規化された論理式である。

$$\begin{aligned}
qual &:: = C_1 \wedge C_2 \wedge \cdots\cdots \wedge C_n \\
C_i &:: = rf_i \mid jf_i \qquad (j=1, \cdots, n) \\
rf_i &:: = rp_{i1} \vee \cdots\cdots \vee rp_{ih_i} \qquad (h \geq 1) \\
jf_i &:: = jp_{i1} \vee \cdots\cdots \vee jp_{iq_i} \qquad (q_i \geq 1) \\
rp_{ij} &:: = exp_i \quad \theta \quad v \\
jp_{ij} &:: = exp_i \quad \theta \quad exp_j
\end{aligned}$$

ここで，$\theta \in \{<, \leq, =, \geq, >, \neq\}$であり，vは定数である。$rp_{ij}$は，制限述語であり，$rf_j$は，同一の組変数に関する制限述語の和である。

jp_{ij}は，結合述語であり，jf_iは，変数に関する結合述語の和で，結合式と呼ばれる。rf_iは変数g，jf_iは変数g_j，g_hに関する式とすると，各々は次の様に考えられる。

$$rf_j : G \rightarrow \{true, false\} \quad (G = dom(\Omega_G))$$

$$jf_i : G_j \times G_h \rightarrow \{true, false\} \quad (G_j = dom(\Omega_{G_j}),\ G_h = dom(\Omega_{G_h}))$$

算術式exp_iは，集積関数（aggregate function）を含んでもよい。以上より条件式qualは次の様に考えられる。

$$qual : G_1 \times \cdots\cdots \times G_l \rightarrow \{true, false\}$$

ここで　$G_i = dom(\Omega_{G_i})$　$(i=1, \cdots, l)$

GC関係$G_1, \cdots, G_l$に対する問合せの意味は，次の様である。

$$T = \{t \mid t = tl(g_1, \cdots, g_l)\} \bigwedge_{i=1}^{l} g_i \in G_i \wedge qual(g_1, \cdots, g_l)\}$$

ＪＤＤＢＳでは，ＧＣ問合せに以下の制限を設ける。

(1) 結合式は，全て等結合だけから成るとする。

(2) group - by集積関数及び単純集積関数内に，group - by 集積関数を含んではならない。

(3) 問合せは算術関数（たとえば，exp，log，sin ）を含まない。

ＧＣデータ構造定義

ＧＣデータ構造は，各データモヂュールＤＭ$_i$のＬＣデータ構造Ｌ$_i$（$i=1, \cdots, n$）上の視野として定義される。よって，ＧＣデータベースｇは，仮想的となる。ここで，ＬをＬＣデータ構造の集合とする（Ｌ＝{Ｌ$_1$, ⋯, Ｌ$_n$}）。ＬをＬＣスキーマの集合として（$\mathscr{L}=\{\underline{L}_1, \cdots, \underline{L}_n\}$）をＬＣデータベースの集合とする（$\mathscr{L}=\{\mathscr{L}_1, \cdots, \mathscr{L}_n\}$）。ＧＣデータ構造上の定義式$\delta$は，$\Omega_{R_{ij}}$（$L_{ij} \in \mathscr{L}_i$）によって表され，$\delta$は，$\mathscr{L}_1 \times \cdots\cdots \times \mathscr{L}_n$からＧへの関数である。ＧＣデータ構造内の各ＧＣ関係スキームは，次のＧＣデータ構造定義言語（ＧＣＤＬ）によって定義される。

range （l_1, L_1 : d_1）⋯⋯（l_p, L_p : d_p）；⋯⋯(1)

define G（a_1, ⋯, a_m） ＜def＞；⋯⋯⋯⋯⋯(2)

ここでＬ$_i$は，ＬＣスキーマd_i内のＬＣ関係名で，l_iはその組変数である。組変数の定義域は，ＬＣ関係Ｌ$_i$で，値としてＬ$_i$内の組を取る。(1)のrange文は，ＬＣ関係Ｌ$_i$のＬＣ組変数l_iを定義している。（$i=1, \cdots, p$，$p \geqq 1$）。

(2)のdefine文は，ＬＣ関係の直積Ｌ$_1 \times \cdots\cdots \times$ Ｌ$_p$から，スキームG（a_1, ⋯, a_m）なるＧＣ関係Ｇへの関数を定義している。＜def＞は，次の様である。

＜def＞ ::=＜subdef＞|＜subdef＞ ＜sop＞ ＜def＞|(＜def＞)

＜sop＞ ::= U | ∩ | −

＜subdef＞ ::= tl where qual

::=（$a_1=exp_1$, ⋯, $a_m=exp_m$）

ここで，exp_iは，定数，ＬＣ属性又はＬＣ属性上の算術式である。

tlは，関数$L_1 \times \cdots \times L_p \rightarrow G_i$（　G）と考えられる。qual は，次の様に積正規化された論理式である。

qual :: $= C_1$ and …… and C_h

C_i :: $= p_{i1}$ or …… or p_{il_i} （$i=1, \cdots, h$）

p_{ij} :: $= \exp_i \ \theta \ \exp_j$

ここで，$\theta \in \{<, \leq, =, \geq, >, \neq\}$である。expは，定数，ＬＣ属性，又はＬＣ属性上の算術式である。C_i 内の各 p_{ij}（$j=1, \cdots, l$）は，皆同一のＬＣ変数を参照してはいるものとする。p_{ij} が $\exp \theta$ v又は，$\exp_1 \theta \exp_2$（ここで$\exp_1$と$\exp_2$は，あるＬＣ変数lのみを参照している）であるとき，制約述語と呼び $p_{ij} : L \rightarrow \{$ true, false$\}$（$L=\mathrm{dom}(\Omega_L)$）なる述語である。又，p_{ij}が，$\exp_1 \theta \exp_2$で，$\exp_1$と$\exp_2$の参照するＬＣ変数が各々，l_1とl_2で$l_1 \neq l_2$のとき，これを結合述語$L_1 \times L_2 \rightarrow \{$ true, false $\}$（$L_i = \mathrm{dom}(\Omega_{L_i})$）と呼ぶ。又，各expは集積関数を含んでもよい。集積関数については第Ⅳ部の２章を参照されたい。

以上，qual は，述語$L_1 \times \cdots\cdots \times L_p \rightarrow \{$ true, false $\}$と考えられる。

よって，１つの subdef の意味は，$G_i = \{g_i \mid g_i = \mathrm{tl}(l_1, \cdots, l_p) \wedge (l_1, \cdots, l_p) \in L_i \times \cdots \times L_p \wedge \mathrm{qual}(l_1, \cdots, l_p)\}$となる。

(1)の定義式は，各 subdef のG_iに，集合演算$< \mathrm{sop} > \in \{\cup, \cap, -\}$を適用した結果を，$\underline{G}(a_1, \cdots, a_m)$のＧＣ関係とすることを表している。

分散情報（ＤＩ）

分散情報（ＤＩ）は，以下の情報から成る。

(1) ＧＣデータ構造Ｇの定義

(2) ＧＣデータ構造と，ＬＣデータ構造との対応関係

2.2.3 分散トランザクション処理

2.2.3.1 分散問合せ処理

分散問合せ処理（ＤＱＰ）とは，全体概念（ＧＣ）データベース上の検索演算を，ローカル概念（ＬＣ）データベース上の演算によって表し，各ＤＭ内／ＤＭ間の通信処理によって解を求めることである。ＧＣデータ構造上の演算をＧＣ問合せ（ＧＱ）とし，ＬＣデータ構造上の検索演算をＬ

C問合せ（ＬＱ）とする。ＤＱＰは，次の３つの主要なステップより成る。

(1) ＧＣ問合せ（ＧＱ）を，ＬＣ関係を参照する問合せで表す。これを全体ＬＣ問合せ（ＧＬＱ）と呼び，この処理を表現変換（ＥＴＰ）と呼ぶ。

(2) ＧＬＱを，各データモジュール（ＤＭ）で独立に処理できる部分をＬＣ問合せ（ＬＱ）として分割し，対応するＤＭでＬＱを実行させる。これを初期ローカル問合せ処理（ＩＬＱＰ）と呼ぶ。

(3) ＩＬＱＰの後のＧＬＱは，ＤＭ間の結合から成る。これをＤＭ内／間の通信処理によって処理して，ＧＱの解を得て，利用者に出力する。これを，ＤＭ間問合せ処理（ＤＭＱＰ）と呼ぶ。

ＥＴＰ

ＥＴＰでは，ＧＣＳ関係に対するＧＣＳ問合せを，ＬＣＳ関係を参照する全体ＬＣＳ問合せ（ＧＬＱ）に変換する。ＧＣＳ問合せ（ＧＱ）も，ＧＬＱもともに，ＲＱＬにより記述される。

ＧＱからＧＬＱへの変換は，ＤＩのＧＣＳ関係式を用いて，問合せ修正（guery medification）〔STONM76〕手法を用いて行われる。

ＩＬＱＰ

ＧＬＱを入力として，ＧＬＱを各ＤＭで独立して処理できるＬＣＳ問合せに分割する。これを問合せ分割と呼ぶ。ＤＭで独立して処理できる部分として，次のものがある。

(1) 同一のＤＭ内の関係間の結合

(2) 関係に対する制限

(3) ＧＬＱの目標属性とＤＭ間の結合属性についての射影。

ＩＬＱＰ〔ＨＥＶＮＡ７８〕は，後述するＤＭ間問合せ処理（DMQP）における通信量を減少させることも目的であるが，異種のＤＤＢＳではむしろ，異種のデータ構造から必要なデータを関係として，導出する目的をもっている。

ＤＭＱＰ

データモジュール（ＤＭ）間問合せ処理（ＤＭＱＰ）とは，初期ローカル問合せ処理（ＩＬＱＰ）後のＤＭ間問合せ（ＤＭＱ）を，問合せ処理に必要なデータを有するＤＭ内及びＤＭ間の通信処理によって解くことである。ＪＤＤＢＳでは，Ethernet，無線網等の放送通信が可能な網を利用している。これにより，効率のよい処理が可能である。これは2.3のＬＩＳ

においてもう少し詳しく述べる。

2.2.3.2　分散更新処理

分散更新処理（Distvibuted update processing）とは，複数のデータモジュール（DM）内のデータに対する更新を行うトランザクションを，DB間の通信処理と，DM内のローカル更新によって処理することである。分散更新処理は，複数のもとで，以下の目標を達成する必要がある。

(1) DM間のデータベースのインテグリティ（DDBS全体のインテグリティ）を保つ。

(2) なるべく多くのトランザクションが同時実行できるようにする。

コミットメント制御

(1)のインテグリティを保つために，コミットメント制御を用いる。トランザクションが複数のデータオブジェクトを更新する際に，これらのオブジェクトへの更新が原子的（atomic）になされることを保障するのが，コミットメント制御の役割である。即ち，各トランザクションTが，オブジェクト x_1，……，x_n（$n \geqq 1$）に対して更新を行うとき，Tによって x_1，……，x_n への更新が全て行われたか，まったく行なわれなかったかのどちらかにしなければならない。コミットメント制御の方法としては，2相コミットメント方法が有名であるが，制御の手数が多く，一般的には低効率となりやすい。ここでは，放送通信網を用いた，より効率的な放送2相コミットメント制御を提案している。これは，2.3のLISにおいて，もう少し詳しく述べる。

同時実行制御

同時実行制御は，複数のトランザクションを同時に実行させるものである。この時，データベースの状態を無矛盾に保つことが必要で，かつ効率のよい制御を考えなければならない。コミットメント制御に，ここでは放送通信を用いた高信頼かつ高効率なものを提案している。これも2.3のLISにおいて詳しく述べる。

2.3 ローカル情報システム（ＬＩＳ）の概要

ここではローカル情報システム（ＬＩＳ）の論理構造とアーキテクチャについて述べる。

ＬＩＳの論理構造は基本的にはＪＤＤＢＳの４層構造である。したがってＬＩＳはＪＤＤＢＳのミニ版と見ることができる。すなわち，次の目標を設定している。

(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

統分化された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フェースを提供する。

(4) 安全性（セキュリティ）の管理

個人ベースに高機能のワークステーションを持つようになると，機密データの漏えいが問題となる。

そこで，ここでは，2.2で詳しく説明しなかった，分散トランザクション処理の概要を中心として，ＬＩＳの論理構造，アーキテクチャについて述べる。

2.3.1 ＬＩＳの論理構造

ＬＩＳの論理構造は基本的にはＪＤＤＢＳと同じである。したがってここでは要約して述べる。まずデータモデルは，データ構造と操作演算とで定まるとする。データ構造は，時間不変な論理構造を与えるスキーマと，これに実際のデータを与えたデータベースとから成る。操作演算には，ある目標スキーマを満足する目標集合をデータベースから導出する検索演算と，データベースを変化させる更新演算とがある。操作言語を表す言語をデータ操作言語ＤＭＬとする。実際に実現されたデータモデルには，関係，網（CODASYL）階層の三種の型がある。各型を，スキーマとＤＭＬの対によって特性化する。

ＤＢＳとはある型のデータモデルを利用者に提供する処理システムである。ＤＢＳの異種性を，各データモデルの型（即ち，スキーマとＤＭＬ）の差と

する。ここでは異種ＤＢＳの同種化とは，各ＤＢＳのデータ構造の集合に対して，これと等価な一つのデータ構造とＤＭＬとを利用者に提供することであり，統合化とは，各ＤＢＳのやデータの所在を意識させない利用形態にすることであるとする。

2.3.1.1　同　種　化

異種ＤＢＳの同種化とは，各ＤＢＳの実際のデータモデルに対して，これと等価な共通の型のデータモデルを利用者に提供することである。2.2では各ＤＢＳに対して，次の結果を示した。

(1) 等価な関係型のローカル概念（ＬＣ）モデルのデータ構造が存在する。

(2) この上の述語論理形式のＲＱＬ操作演算を満足する目標集合を導出するＤＢＳ操作演算が存在する。

従って，従来のＤＢＳ上にＬＣモデルを提供するインタフェースを設けることができる。この様にして同種化されたＤＢＳを仮想ＤＢＳとする。

仮想ＤＢＳを通信網で結合したシステムを，ローカル概念層のＤＤＢＳとする。

2.3.1.2　統　合　化

仮想ＤＢＳの統合化とは，各点にある仮想ＤＢＳのデータ構造の集合に対し，これと等価なデータ構造と操作演算とを全利用者に提供することである。ここでは，各仮想ＤＢＳ内のデータ構造そのものの集合を全体概念データ構造とを定め，これに作用する操作演算を全体概念操作演算とする。

この両者の対で定まるモデルを全体概念データモデル（ＧＣモデル）とする。ＧＣモデルが利用者に提供される階層をＤＤＢＳの全体層とする。

ここで利用者には，ＧＣモデルを提供する１つのＤＢＳが存在することになり，ＤＤＢＳは統合化されたことになる〔図2.10〕。このためには仮想ＤＢＳ間の有効な通信処理システムの設計が必要である。これについては2.3.2で述べる。

次にＧＣモデルの操作演算と，全体トランザクションについて述べる。

全体操作演算･

ＧＣモデルの操作演算は検索と更新演算がある。

ａ．検索演算

全体データベース上の検索演算は，ＲＱＬ〔ＴＡＫＩＭ８３，８４〕で次式の形式で表現される。

var （x_1，X_1）……（x_ℓ，X_ℓ）；……(1)

get into G(Λ) where Ψ；……(2)

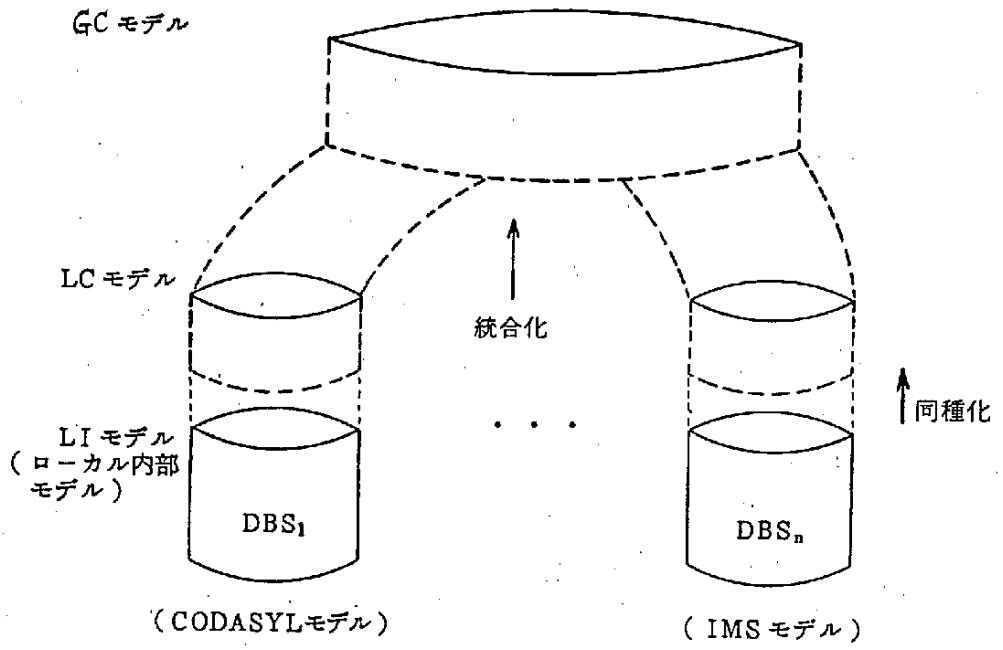


図 2.10　ＬＩＳの論理構造

X_i は全体データベース内の任意の関係で，x_i は X_i 内の組を値としてとる組変数である。(1)式は，X_i に対して組変数 x_i を定める。Ψ は検索条件式で，(i) x_i．a θ v と(ii) x_i．a θ x_j．bの形式の二重の基本式の積・和で与えられる。x_j．aは，x_i のとる組の属性aの値を表す変数である（x_j．bも同様）。θ は比較演算子，vは定数である。Λ は目標関係Gのスキームを与える目標リストで，x_i．aの形式の変数の並びである。

全体検索演算のＲＱＬ表現を全体問合せ（Λ，Ψ）とする。この意味を次式で与える。

$G = \{ t \mid t = \Lambda(v) \wedge \Psi(v) \wedge v \in X_1 \times \cdots \times X_\ell \}$ ……(3)

b．更新演算

全体データベースの更新演算としては，関係Xに対する消去，追加，更新演算がある。これらの演算は，ＲＱＬで次の様に表現される。以下，Y_1，…，Y_m は，全体概念データベース内の任意の関係とする。

(a) 消去演算

var （ x, X）（ y_1, Y_1 ）…（ y_m, Y_m ）；……(4)
del　x　where　Ψ ；

は，x，y_1，…，y_m 上の条件式である。この意味は，Xから，$t \in X \wedge w \in Y_1 \times \cdots \times Y_m \wedge \Psi(t, w)$ なる組tを除くことである。

(b) 追加演算

var （ y_1, Y_1 ）…（ y_n, Y_m ）；……(5)
app　X（ Λ ）where Ψ ；

Ψ と Λ は，y_1，…，y_m 上の目標リストと条件式である。この意味は，関係Xに，$w \in Y_1 \times \cdots \times Y_m \wedge \Psi(w) \wedge t = \Lambda(w)$ なる組tを加えることである。

(c) 置換演算

var （ x, X）（ y_1, Y_1 ）…（ y_m, Y_m ）……(6)
rep　x（ Λ ）where　Ψ

Λ と Ψ は，x，y_1，…，y_m 上の目標リストと条件式である。この意味は，$t \in x \wedge w \in Y_1 \times \cdots \times Y_m \wedge u = \Lambda(t, w) \wedge \Psi(t, w)$ なる組tをuで置換することである。

全体トランザクション

全体トランザクションとは，全体概念データ構造に対する組単位の検索（ read ）演算と更新（ write）演算とで定まる系列である。単に，全体トランザクションは，原子的な実行単位である。即ち，全体トランザクションは，正しく実行されるか，全く実行されないかのどちらかである。

全体トランザクションは，正しい全体概念データベースを，正しい全体概念データベースに変化させるが，全体トランザクションの各演算の実行段階では，必ずしも全体概念データベースの状態は正しくない。

全体トランザクションの操作演算は，全体概念データ構造内の関係を組単位に操作する。操作演算は，各関係X毎の現在子（ currency　又は，cursor ）x に基づいている。x は，関係X内の組を指示している。関係

Xに，複数の現在子を設けられる。以下に，基本操作演算を関係として与える。

(1) var（X）;

input X　　＝全体概念データ構造内の関係名

output var　＝関係Xの現在子（へのポインタ）

例　crtx 1＝ var（X）;

crtx 2＝ var（X）;

crtx 1とcrtx 2を，関係Xの現在子とする。

(2) {first / last}（crtx）;

input crtx ＝関係Xの現在子

output {first / ast}　OK if 成功（Xの現在子）

NO if 失敗（Xには組がない）

関係Xの現在子crtx を，Xの格納順の先頭（first）又は最後（ast）の組を指示させる。

(3) {next / prior}（crtx）;

input crtx ＝Xの現在子

output next / prior ＝ OK if 成功

NO if 失敗（Xには組がない）

関係Xの現在子crtx の指示する組で，一つ次（next）又は，一つ前（prior）の組とする。組がないときにはNOがかえり，crtx の内容はＮＩＬとなる。

(4) read（crtx）;

input crtx＝Xの現在子

output crtx＝{OK if 成功 / NO if 失敗}

crtx の指示する組を，現在子内に読み出す。このとき，Xの任意の属性aに対して，以下を定める。

crtx. a ＝今読まれた組の属性aの値。

＊crtx ＝現在子の指す組の識別子

(5) ertx.a＝v；

現在子crtxの属性aの値をvとする。

(6) del（ertx）；

input　crtx＝Xの現在子

output　del＝ OK if 成功 / NO if 失敗

現在子ertxの指す組をXから除く。＊ertxはＮＩＬとなる。

(7) app（crtx）；

現在子crtx内の組の値を，関係Xに加える。＊crtxは，加えられた組の識別子となる。

(8) rep（crtx）；

現在子crtxの指す組の値を，crtx内の値に置き換える。

例　x＝var（X）；　first（x）；

x.ename＝“鈴木”；　x.age＝28；

rep（x）；

Xの先頭の組のenameを鈴木に，ageを28にする。

(9) abort（）；

トランザクションを異常終了（アボート）させる。

(10) commit（）；

トランザクションをコミットさせる。

(11) begin（）；

トランザクションを開始させる。

(12) end（）；

トランザクションを終了させる。

2.3.2　ＬＩＳのアーキテクチャ

2.3.2.1　システム構成

ＬＩＳの構成としては，各サイトに仮想情報システム（ＶＩＳ）（仮想ＤＢＳ（ＶＢＳ）と同じ）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），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

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

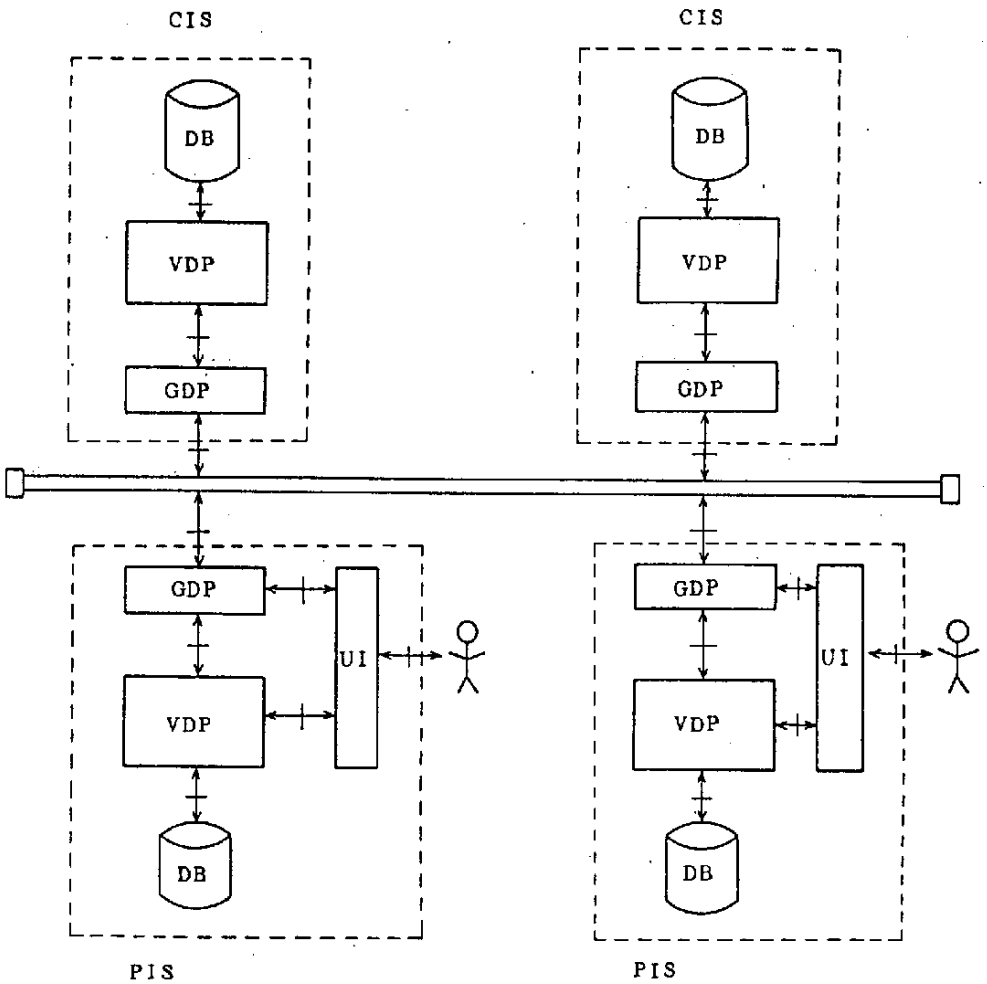


図2.11　ＬＩＳのアーキテクチャ

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2.11〕。

① ＶＤＰ

ＶＤＰは，利用者にローカルなデータに対する関係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〔ＴＡＫＩＭ80，82〕により，関係型とみせることが可能である。（2.2参照）

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。

(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。

2.3.2.2　放送網と群サービス

前述した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2.12 に示す階層構造を有している。トランスポート群制御（ＴＣＣ）層は，ＥＤＬサービスを用いて，高信頼な放送サービスを提供する。分散データベース（ＤＤＢ）層はＴＣＣ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。

ＴＣＣ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。

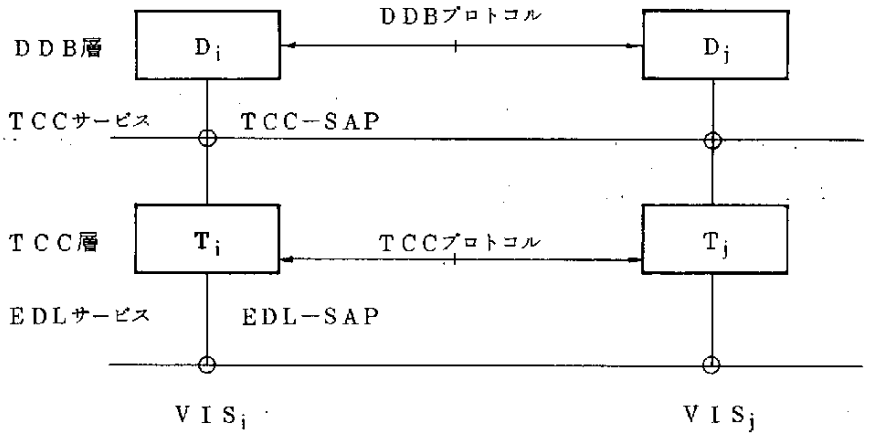


図2.12　ＧＤＰの通信階層

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

情報は出した順番で確実に届く。ＴＣＣのプロトコルについては３章で詳しく述べられる。

ＤＤＢ層はＴＣＣサービスの放送性を用いて，問い合せの最適化，コミットメント制御の最適化の処理に対して，効率的な手法を有している。

2.3.3 ＬＩＳの分散トランザクション処理

この節では，ＬＩＳの分散トランザクション処理について述べる。最初にトランザクションのモデルを定義し，ＬＩＳで採用している同時実行制御，コミットメント制御について解説する。

2.3.3.1 分散トランザクションモデル

同時実行制御を考えるうえでのシステムの論理的なモデルを考える。論理的なシステムは，図2.13に示すように，トランザクション管理モヂュール（ＴＭ）と，１つ以上の仮想データベース・システム（ＶＢＳ）とから構成され，これらは通信網（ＣＬ）で結合されている。

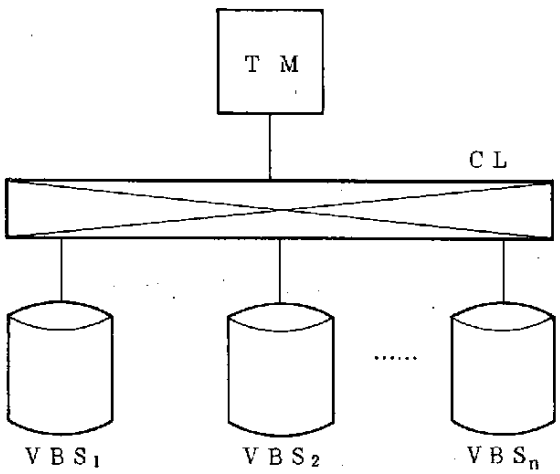


図2.13 ＬＩＳの論理構成

通信網は，2.3.2.2で示す高信頼，高性能放送通信網である。即ち，ＴＭとＶＢＳは，高使頼放送通信のためのプロトコルＴＣＣＰを用いて，以

下の特徴をもった通信を行える。

(1) ある一定のＴＭ，ＶＢＳの集合を一つの群として，設定できる。

(2) 群内のある実体（ＴＭ又はＶＢＳ）が送信したＰＤＵ（プロトコルデータ単位）は，紛失することも，冗長化することもなく，送信順に，群内の全実体に届く。

(3) 群内で，複数の実体がＰＤＵを送信しているとき，どの実体も同一の順序で，送信されたＰＤＵを受信する。

(4) 群内の全実体間で，ＨＤＬＣ手順のように，交互監視方式により，ＰＤＵを送信しあうので，効率がよい。

さて，こうした通信網に接続されているＴＭとＶＢＳの概要を示そう。

まず，ＴＭは，利用者に対するインタフェースである。利用者は，データの所在を独立に，トランザクションＴを記述し，これをＴＭに入力する。ＴＭは，Ｔを各VBS_i毎のローカル・トランザクションT_iに分割し，各VBS_iに送る。VBS_iは，ローカル・トランザクションT_iに基づいて，VBS_i内のオブジェクトの操作を行う。

2.3.3.2 仮想ＤＢＳのモデル

仮想ＤＢＳ（ＶＢＳ）は，図2.14に示すように，オブジェクトの集合としてのデータベース（ＤＢ）と，主記憶（Ｍ）とから構成される。主記憶Ｍは，複数のスロットｓ から構成される。

$$M = \{ s_i \mid i = 0,\ 1,\ \cdots,\ m-1 \}$$

データベースＤＢは，オブジェクトo_iの集合である。

$$DB = \{ o_i \mid i = 0,\ 1,\ \cdots;\ n-1 \}$$

各オブジェクトo_iはオブジェクト識別子（oid），安全性クラス，及び値（属性値の組）とからなる。ここで，xをオブジェクト識別子したとき，

object（x）＝oidがxであるオブジェクト

sc（x）　＝object（x）の安全性クラス

val（x）　＝object（x）の値

という記法を用いることにする。

ここで，主記憶Ｍ内の各スロットs_iには，名前が与えられて，この名前を変数ということにする。ｔを変数としたとき，slot(t)は，ｔの指示するスロットを示すとする。

このとき，仮想ＤＢＳは，次のような基本操作演算を備えている。〔図2.14〕

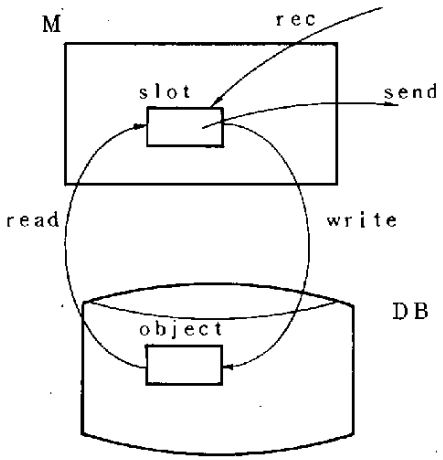


図2.14　ＶＢＳ基本操作演算

〔ＶＢＳの基本操作演算〕

(1)　read（x；t）；

object（x）の値val（x）を，変数tの指示するスロットslot(t)に複写する。

(2)　write（f（t_1，…，t_n）；x）；

slot(t_1)，…，slot(t_n)内の値にある関数f（例えば，加算）を適用した結果を，object(x)のval(x)に書く。

(3)　send（f（t_1），…，slot(t_n)）；

slot(t_1)，…，slot(t_n)内の値に関数fを適用した結果を，通信網に送出する。

(4)　rec（t）；

通信網から，メッセージが届くのを待ち，受信したならば，slot(t)に記憶する。□

以上が，ＶＢＳの基本操作演算である。ＤＢは，オブジェクトの集合であるが，オブジェクトは普遍な存在である。即ち，消滅することも，新たに生成されることもない。ただ，オブジェクトの値と安全性のクラスが変

わるだけである。例えば，データの消去は，オブジェクトの値を空にしてしまう，即ち，空値をwrite 演算で書くことである。又，データの追加は，値が空なオブジェクトに，値を書くことである。

2.3.3.3　ローカル・トランザクション

仮想ＤＢＳ，ＶＢＳ$_i$に対するローカル・トランザクションT_iは，ＶＢＳ基本演算の系列である。

〔定　義〕

ＶＢＳ　Ｖに対するローカル・トランザクションＴは，ＶについてのＶＢＳ基本操作演算を要素とした全順序集合（Ｏ，＜）である。ここで，Ｏは，ＶＢＳ操作演算の集合，＜$\subseteq O^2$は，全順序関係である。□

即ち，ローカル・トランザクションＴは，逐次実行されるプログラムに対応している。

ここでは，Ｔは条件分岐 if－then－else，及び繰返しwhile－do といった制御構造を含まないものと仮定する。

次に全体トランザクションについて考えよう。利用者に対して，システムは図 2.15 に示すように，仮想ＤＢ（ＶＢ）と，仮想主記憶（ＶＭ）とから成る一つの仮想ＤＢＳとみれる。ＶＢは，システムの全オブジェクトの集合である。

$$VB = \bigcup_i DB_i$$

仮想主記憶ＶＭは，スロットの集合である。このとき，ＶＢＳ基本操作演算のなかで，read とwrite が，ＶＢとＶＭについて，同様に定義される。

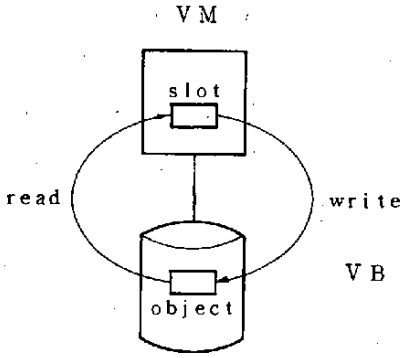


図　2.15

次に，利用者がシステムに与える全体トランザクションを定義してみよう。

〔定　義〕

全体トランザクションTは，仮想ＤＢ（ＶＢ）内のオブジェクトに対するreadとwriteの演算を要素とする全順序集合（O，＜）である。Oは，readとwrite演算であり，$<\subseteq O^2$ は全順序関係である。□

全体トランザクションTでの基本操作演算の順序関係＜は，逐次実行の制御構造を与えている。しかし，分散システムでは，複数の処理装置による並列処理が可能となることから，T内の基本操作演算の順序のなかで意味のあるものだけ，この順に実行させ，他のものは，なるべく並列に実行できることが望ましい。このために，意味のある順序とは何かをを考えてみよう。

まず，二つの基本操作演算間の競合関係を定義しよう。

〔定　義〕

opとop′を二つの異った基本操作演算とする。このとき，以下の条件を満足するならば，opとop′は互いに競合するという。

即ち，opとop′は，同一のオブジェクトに対する演算で，少くとも一方はwriteである。□

〔定　義〕

T＝（O，＜）を全体トランザクションとする。このとき，O上の順序関係$\ll\subseteq O^2$を次のように定義する。

任意の基本操作演算op，op′∈Oに対して，

$$op \ll op' \quad \underline{iff} \quad op \rightarrow op' \text{ 又は } op \cdots\!\!\rightarrow op'$$

ここで，関係$\rightarrow\subseteq O^2$，$\cdots\!\!\rightarrow\subseteq O^2$ は次のように定義される。

$op \rightarrow op'$ iff (i) Tで，$op<op'$であり，
(ii) opとop′は競合する。

$op \cdots\!\!\rightarrow op'$ iff (i) Tで，$op<op'$であり，
(ii) $op = read(x;t)$で，
$op' = write(f(t_1, \cdots, t_n);y)$であり，
$t \in \{t_1, \cdots, t_n\}$である。

以上で，≪を意味のある順序関係といい，→を競合フロー関係，$\cdots\!\!\rightarrow$を

情報フロー関係という。□

全体トランザクションT＝（O，＜）に対して，意味のある順序関係≪により，演算集合を半順序化した（O，≪）を，意味のあるトランザクションT*とする。T*は，次のようにグラフ表現できる。

〔定　義〕

意味のあるトランザクションT*＝（O，≪）に対する次のグラフを，トランザクション・グラフ（又はTグラフ）G＝（N，E）とする。

(1) Nは節点の集合である。各節点は，O内の各演算を示している。

(2) Eは，節点間の有向辺である。opとop′を夫々，O内の演算opとop′を示す節点とする。

このとき，op≪op′ならば，xからyへの有向辺を設ける。有向辺には，≪の特性により，次の二種がある。

(i) 競合フロー辺　　op→op′のとき，

(ii) 情報フロー辺　　op⇢op′のとき。

ここでopとop′が異ったVBS内のオブジェクトを操作しているとき，VBS間情報フローといい，op⇒op′と示す。□

例を考えてみよう。図2.16に示すように，データのオブジェクトxとz，y，vがあり，夫々異なった仮想DBS（VBS）VBS_1，VBS_2，VBS_3にあるとする。このとき，次のような全体トランザクションTがあるとする。

```
T；  read  ( x ; a ) ;
     read  ( y ; b ) ;
     write ( a ; z ) ;
     read  ( v ; c ) ;
     write ( a + c ; v ) ;
     write ( b ; x ) ;
```

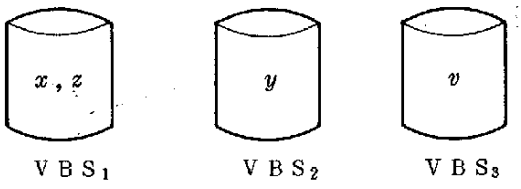


図2.16　オブジェクトx，y，z，vの配置

ここで，a，b，cは変数である。このTに対するトランザクション(T)グラフGを2.17に示す。

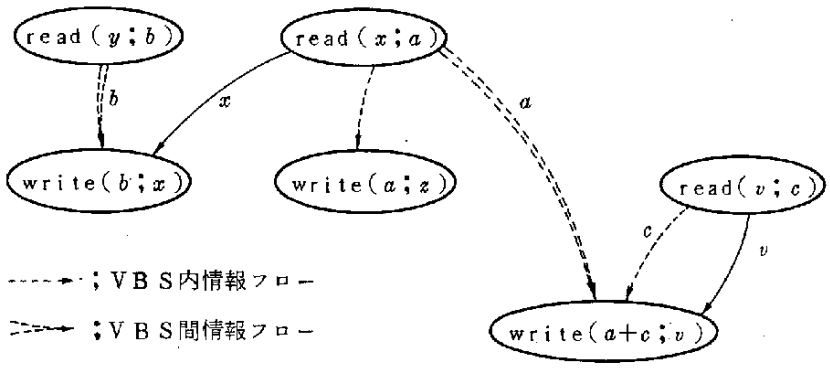


図2.17　Tグラフ

2.3.3.4　トランザクション分割

トランザクションの分散処理モデルとしては，サーバ・モデルとプロセス・モデルの二つがある。〔図2.18,2.19〕

サーバ・モデルでは，利用者のいるワークステーション(WS)から，全体トランザクションTの演算順序に従って，演算opを，そのオブジェクトを有するVBSに送出し，結果を受信しながら，処理を進めていく。即ち，WSのハンドシェイキングが行われる。一方，プロセス・モデルでは，トランザクションを，各VBS毎の部分トランザクションに分割し，各VBSで分散実行される。

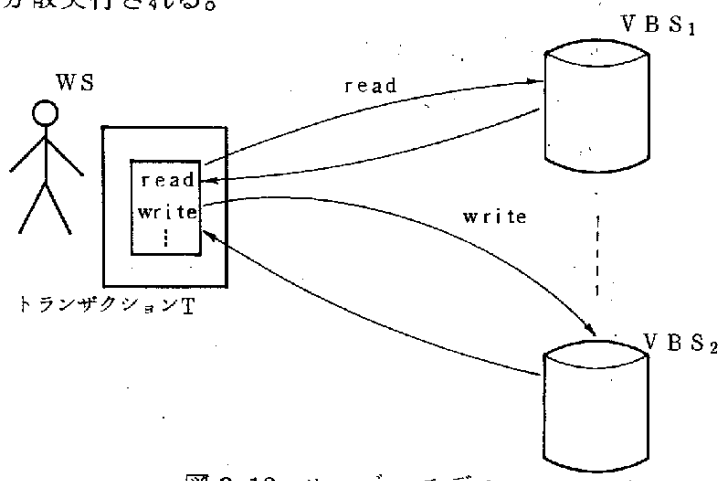


図2.18　サーバ・モデル

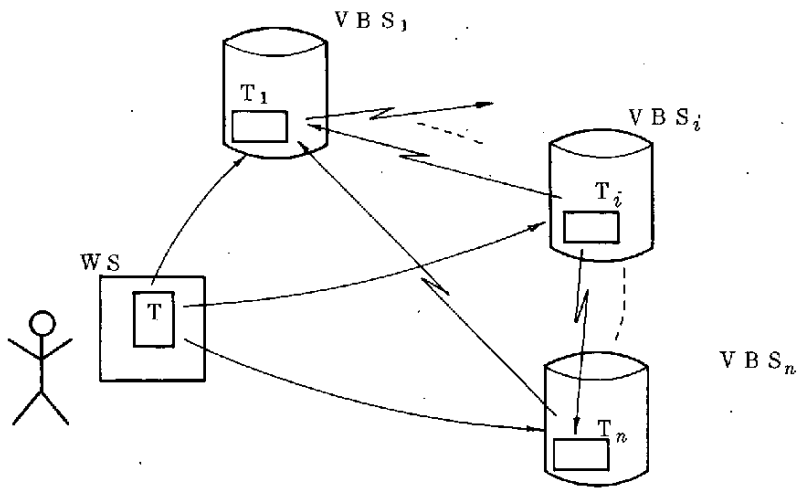


図2.19　プロセス・モデル

サーバ・モデルでは，WSとVBS間で通信が行われるが，プロセス・モデルではVBS間で通信が行われる。サーバ・モデルでは，VBS間での情報フローを，WS経由で行う必要がある点と，WSからのハンドシェイクにより処理が進められるので，通信負過が高くなる点が問題となる。

このために，LISでは，プロセス・モデルを用いている。

利用者からの全体トランザクションTは，Wで，各VBS_i毎の部分トランザクションT_iに分割される。各T_iは，VBS_iで実行される。まず，全体トランザクションTの分割方法を与える。

〔全体トランザクション分割方法〕

全体トランザクションTに対して，トランザクション・グラフGをつくる。

(1) G内の各VBS間情報フロー辺 read（x；t）⟹write（f（t_1，…，t_n）；y）に対して，

(i) この情報フロー辺をGから除く。

(ii) 新たに，節点 send（t；m）と rec（m；t）をGに加え，VBS内情報フロー辺

read（x；t）……> send （t；m）

rec（m；t）……>write （f（t_1，…，t_n）；y）

をGに加える。

(2) G内の連結な部分グラフG_1, …, G_n を求める。各G_iに対して，ローカル・トランザクションT_iをつくる。□

以上の分割手続きで，各G_iは，あるＶＢＳ内のオブジェクトのみを操作するものであることは明らかである。ここで，分割された部分トランザクションT_iが操作するオブジェクトの存在する仮想ＤＢＳをＶＢS_iとする。全体トランザクションから分割された部分トランザクションT_1, …, T_nは，夫々，ＶＢS_1…，ＶＢS_nへ送信され，実行される。ＶＢＳ間の通信は，部分トランザクション内のsendとrec演算により行われる。

図2.17の全体トランザクションＴの分割例を図2.20に与える。

いま，ここで示した分割方法では各ＶＢS_iがT_iを一定の順序で逐次実行していく場合にはデッドロックが生じる可能性がある。

例えば，図2.21に示す全体トランザクションを考えてみよう。これを図2.22のように部分トランザクションT_1とT_2に分割する。

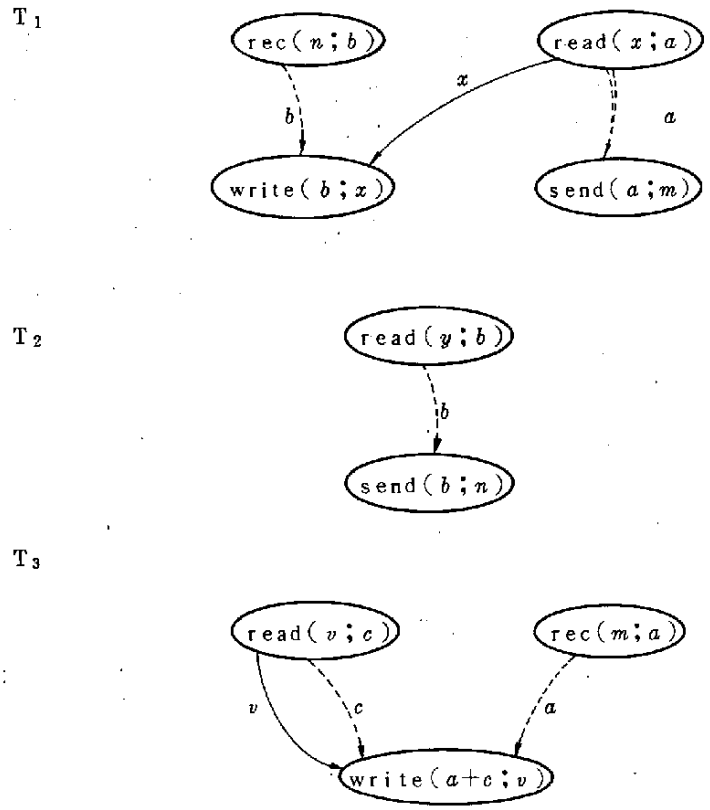


図2.20 部分トランザクション

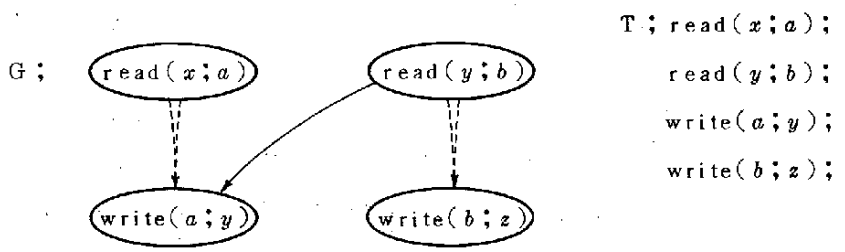


図 2. 21

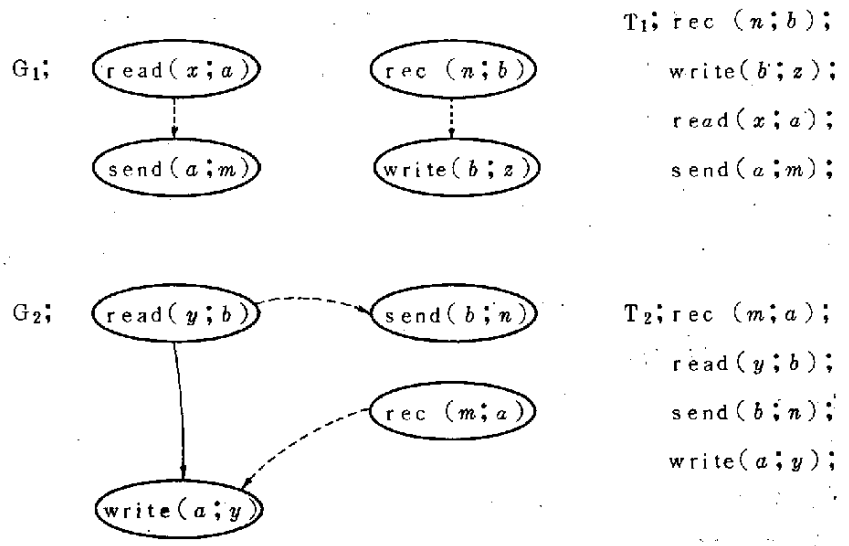


図 2. 22

二つのトランザクションをみてわかるように，最初の rec 演算で，互いに他のトランザクションからのメッセージを待ち合うデッドロック状態におちいってしまう。このためには，次のようにする。問題は，1つの部分トランザクション内の通信演算 send と rec の順序が問題となっている。

1つの解は，与えられた全体トランザクション T＝（O，＜）の全順序関係＜に従って，部分グラフ G_i から T_i をつくることである。このとき，read（x；t）:::> send（t，m）なる send は，read の直後の順番とし，rec（m_j t）:::> write（$f(t_1, \cdots, t_n)$；y）なる rec は write の直前の順位と考える。こうすると，T_1 と T_2 は，夫々，次のようになる。

T_1 ；	read (x ; a) ;	T_2 ；	read (y ; b) ;
	send (a ; m) ;		send (n ; b) ;
	rec (n ; b) ;		rec (m ; a) ;
	write (b ; z) ;		write (a ; y) ;

こうするとデッドロックは生じない。各VBS$_i$では，通信演算の順序をT_iと同じにしておきながら，他の演算は意味のある順序関係を保つ形で，順序のいれ換えを行える。

2.3.3.5　ログと直列可能

次に，プロセス・モデルのログの直列性（ servialization ）について考えてみる。まず，Πを，n個の全体トランザクションT_1，…，T_nの集合とする。

〔定　義〕

$T_{ij}=(O_{ij}, <_{ij})$を，全体トランザクションT_iのVBS$_j$に対する部分トランザクションとする。Π_jを，T_{1j}，…，T_{nj}の集合とする。Π_jに対するログL_jは，全順序集合$(O_j, <_j)$である。ここで，

(1)　$O_j=\bigcup_i O_{ij}$

(2)　$\bigcup_i <_{ij} \subseteq <_j$　である。□

〔定　義〕

Πに対するログLは，半順序集合$(O, <)$であり，

(1)　$O=\bigcup_j O_j$

(2)　$\bigcup_j <_j \subseteq <$　である。□

さて，Πに対するログLが直列であることの意味を定義しよう。

〔定　義〕

全てのL_jで，任意の全体トランザクションT_k，T_ℓに対して，T_k，T_ℓの任意の演算op_k，op_ℓについて，$op_k <_j op_\ell$であるような全順序関係T_1，…，T_nがあるとき，Lは直列であるという。□

〔定　義〕

LとL′をΠに対するログとする。このとき，

(i)　意味のある順序が同じで，

(ii)　各オブジェクトに対する最後のwriteが同じであるとき，

LとL′は同値であるという。□

〔定　義〕

ΠのログLが，Πのある直列ログと同じならば，Lは直列可能であるという。□

以上より，T_iとT_jを二つの全体トランザクションとしたとき，あるVBS$_k$で，競合する演算op$_i$とop$_j$がop$_i$ $<_k$ op$_j$であるならば，他の全てのVBS$_\ell$で，T_iとT_jの競合演算は，T_iのものが先に行われる。これは，従来のサーバ・モデルでの直列性の定義である。LIPは，プロセス・モデルのために，もう一つ各全体トランザクション内で成り立っている。VBS間情報フローの順序関係が，ΠのログLで成り立っていなければならない。

以上から，直列可能なログLを，次のように再定義できる。

〔定　義〕

ΠのログLは，半順序集合（O，$<$）である。このとき，以下が成り立つとき，Lは直列可能であるという。

(1) 各部分ログL_jは，全順序集合（O_j，$<_j$）であり，各$T_{ij}=(O_{ij}, <_{ij})$で，op $<_{ij}$ op′ならば，L_jでop $<_j$ op′。

(2) T_1，…，T_nに対するある全順序関係$\ll$があり，op$_k$とop$_\ell$を夫々T_kとT_ℓのVBS$_j$内のオブジェクトに対する競合演算とすると，$T_k \ll T_\ell$ならば，op$_k$ $<_j$ op$_\ell$である。

(3) 各T_iで，op $\ll_j$ op′ならば，Lで，op $<$ op′である。□

LIPでは，同期方法としては，二相ロック（2PL）方式を用いている。すると，次の定理が成り立つ。

〔定　理〕

任意の全体トランザクションTに対して，Tがトランザクション分割手続で部分トランザクションT_1，…，T_nに分割され，各T_iが2PLトランザクションならば，得られるどのログLも直列可能である。

〔証　明〕

2PLにより，定義の(1)と(2)は保たれることは明らかである。又，(3)は，send-rec 機構により保たれる。□

以上より，LIPでのどのログも正しいことがわかる。

2.3.3.6　同期方式

ＬＩＳでは，各ＶＢＳは，オブジェクト単位のロックによる同期方式を用いている。トランザクションは，２PL形式で，readを行う前にR lock（共有可能ロック），writeを行う前にW lock（排他ロック）を行う。

各部分トランザクションは，全体トランザクションがコミットした時点で，全ロックの解放を行う。コミットメントの方式については，この次に述べる。

ロックによる同期方式の問題点は，デッドロックが生じる可能性があることである。ＬＩＳでは，複数のＶＢＳ間でのデッドロックを検出することは，ＶＢＳ間での通信を行わなければならないため困難であるので，デッドロックが起こらないようにする防止方法を用いている。

いわゆる横取り（preemption）方式である。全ての全体トランザクションを全順序付けるわけであるが，時刻印によるのではなく，各ＶＢＳへの部分トランザクションの到着順とする。これは，３章で述べるように，ＬＩＰの通信システムは，高信頼放送網であり，ある点から送信されたメッセージは，網内の全点で，必ず，正しく，順番通り受信される。又，複数の点が送信している場合も，各点では，同一の順序でメッセージが受信される。従って，WSが，全体トランザクションTから部分トランザクション集合｛T_1，…，T_n｝をつくり，これを放送する。すると，各ＶＢＳは，同一の順序で，複数の全体トランザクションを受信できる。この受信順を全体トランザクション番号に基づいて回避される。まず，xをオブジェクトとし，既に，全体トランザクションT′によりロックされているとし，全体トランザクションTが，xをロックしようとしているとする。ここで，seq(T)を，Tの番号とする。このとき，次の手順を用いる。

```
if   seq(T′) > seq(T)
        then        T′をabout
        else        Tをwait
```

2.3.3.7　コミットメント制御

全体トランザクションTは，全ての部分トランザクションT_1，…，T_nがコミットしたことが確認されたとき，コミットする。このための方式として，各ＶＢＳの分散制御と，高信頼放送通信を利用した二相コミット

（２ＰＣ）方式を用いる。以下にその手続きを与える。〔図2.23〕

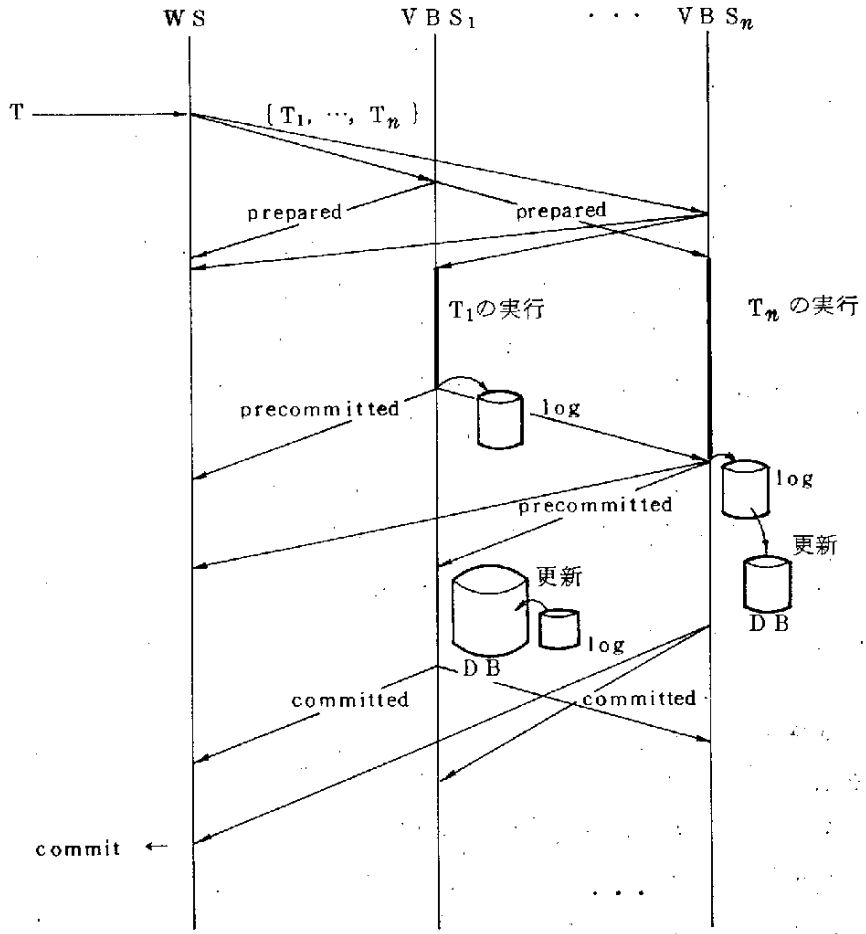


図 2.23　コミットメント制御

〔２ＰＣ手順〕

(1) 全体トランザクションTを，WSは，部分トランザクション$T_1 \cdots, T_n$に分割する。｛$T_1 \cdots$, T_n｝を放送する。

(2) 各VBS_iは，｛$T_1 \cdots$, T_n｝を受信し，T_iが実行可能ならば，preparedメッセージを放送する。

(3) 各VBS_iは，全VBS_jからpreparedを受信したならば，T_iの実行を開始する。

(4) T_iの実行が正常終了したならば，更新データをログに格納し，precommittedを放送する。

(5) 全 precommitted を受信したならば，ログよりＤＢの更新を行う。正しく更新が行えれば，T_i の全ロックを開放し，committed を放送する。

(6) ＷＳ，ＶＢＳ$_i$ は，全committedを受信したならば，コミットする。□

2.4 高水準インタフエース（ＬＩＰ）の概要

論理型言語インタフェースプロセッサ（Logical Interface Processor）は，従来のＤＢＳが提供する操作演算（ＤＭＬ）に無かった機能を加えた，高水準インタフェースを提供するものである。

ＬＩＰの提供する，強力な問合せ言語の特徴としては，次のことがあげられる。

(1) 推論機構により，従来，行ないにくかった問合せが容易になる。（非手続き言語，仮想関係，再帰的定義等）

(2) 利用者が自分に必要なデータベースの視野（ビューView）を柔軟に，かつ容易に定義できる。

2.4.1 論理とデータベース

論理をもちいて，データベースを表現することは，ＰＲＯＬＯＧデータベースを代表として，最近よく行なわれている。その理由としては，

(1) 関係データベースは述語論理を基盤としている。

(2) 従来のデータベースが物理的側面が強いため，より柔軟な構造として扱えるように，概念的統一化が望まれている。

(3) 従来のデータベースが実世界の事実の集合のみであったのに対して，実世界の規則の集合を含めた形に拡張し，効率化及び検策の柔軟性を高めたい。

などがあげられる。

上記のことを実現するためには，論理的な推論機構が必要である。このようにデータベースを論理で表現し，推論機構によって，データを処理するデータベースシステムを演繹データベースシステムと呼ぶ。

ＬＩＰは演繹データベースシステムとは異なるが上記のような理由を目標として，既存のデータベースシステムの機能を拡張するものである。

2.4.2　データベースの論理表現

ＬＩＰではデータベースの事実や，規則を論理式の形式でとらえている。

詳しい説明は第５章に述べ，ここでは非公式な形で論理表現を述べる。

事　実

データベースの中の事実は，論理では，節表現で次のように表わす。
（ ground clause ）

$P(c_1, \cdots, c_n) \leftarrow .$

Pは述記記号，$c_1, \cdots, c_n$は定数

例えば，ある関係を論理表現すれば，次のようになる。

（例）　関　係　　論理表現

R

父	子
和雄	太郎
洋	花子
⋮	⋮

⟹

父（和雄，太郎）←.
父（洋，花子）←.

規　則

実世界の規則は次のように表わす。

$P(t_1, \cdots, t_m) \leftarrow .$　　（t_iの少なくとも一つが変数）

又は，

$R(t_1, \cdots, t_k) \leftarrow . \quad P_1(t_1^{p_1}, \cdots, t_{n_{p_1}}^{p_1}) \wedge \cdots\cdots \wedge P_m(t_1^{p_m}, \cdots, t_{n_{p_m}}^{p_m}).$

（例）　子供の父の妻は母である。

母（x，y）←父（z，y）∧妻（x，z）.

質　問

データベースに対する問い合せは，次の形で表わす。

$\leftarrow P_1(t_1^{p_1}, \cdots, t_{n_{p_1}}^{p_1}) \wedge \cdots\cdots \wedge P_m(t_1^{p_m}, \cdots, t_{n_{p_m}}^{p_m})$.

（例）　太郎の父は誰か？

←父（x，太郎）.

2.4.3 ＬＩＰの推論機構

ＬＩＰでは，論理の推論機構として，反駁機構を用いている。この機構により，データベースの質問処理がなされる。反駁機構にも，多種のものがあるが，ＬＩＰでは拡張されたＳＬＤ反駁を用いている。これは従来のＳＬＤ反駁に比べて，不必要なバックトラッキングを省いている点で，すぐれている。詳しくは５章を参照されたい。

2.4.4 仮想関係と再帰的定義

ＬＩＰでは，仮想関係を作ることや，再帰的な定義が行なえることにより，従来のＤＭＬに比べて，強力なものとなっている。

仮想関係（ Virtual Relation ）

2.4.2の規則の例で出てきた，母という述語の定義がこれにあたっている。つまり，データベースの中には陽に母という関係はないが，あたかも，在るように定義できる。

母（ x， y ）←父（ z， y ）∧妻（ x， z ）.

再帰的定義（ Recursive Definition ）

規則の定義において再帰的な定義が許される。例えば，先祖の定義として，

先祖（ x， y ）← 親（ x， y ）.
先祖（ x， y ）← 親（ x， z ）∧先祖（ z， y ）.

とでき，a の先祖を求めたいという質問に際しては，すべての先祖を求めることが可能である。

視　野（ View ）

利用者が個別に規則を定義することにより，その利用者個有のデータベースの視野をもてる。従来，利用者の視野はデータベースのスキーマやデータに対して，その部分集合の定義の形でのみ許されるものがほとんどであったが，ＬＩＰでは，本来のデータベースをはみ出した形で利用することが可能である〔図2.24〕。

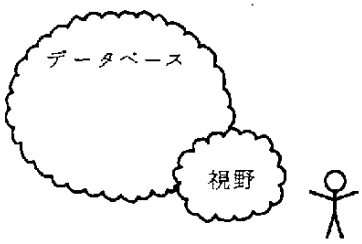


図 2.24　拡張された視野

（例）

社員住所（氏名，住所，電話）

社員趣味（氏名，趣味）

社員年令（氏名，年令，姓別）

の各事実の集合があった時，

例えば，Aさんの視野として，

花嫁候補（n，a，t，h）←社員住所（n，d，t）∧社員年令（n，a，s）∧社員趣味（n，h）∧$a \leqq 25 \wedge s =$ "女".

などが，個人の視野として定義できる。

2.5　実現化

ここではＪＤＤＢＳ，ＬＩＳ，ＬＩＰの実現レベルでの構成について述べる。

2.5.1　ＪＤＤＢＳ

ＪＤＤＢＳは，現在，図2.25に示す様に，当協会のM－３６０Ｒ内の４つのCODASYL DBSから構成されている。各CODASYL DBSはＡＩＭ（FACOM）ＤＢＭＳである。

ＪＤＤＢＳは，通信網（ＣＮ）で結合された４つのデータモジュール（DM）から成る。各DMは，(1)～(4)のCODASYL DBSの１つと，ローカルデータベースプロセッサ（ＬＤＰ），全体データベースプロセッサ（GDP），システム格納領域（ＳＳ）から成る。ＳＳは，分散情報（ＤＩ），異種性情報（ＨＩ），及び関係作業域（ＲＷＳ）から成る。

ＳＳは１つのＡＩＭ　ＤＢＳとして実現されている。ＧＤＰとＬＤＰは独立したプロセッサとして存在していて，各々ＳＳ内のＤＩとＨＩを用いて動

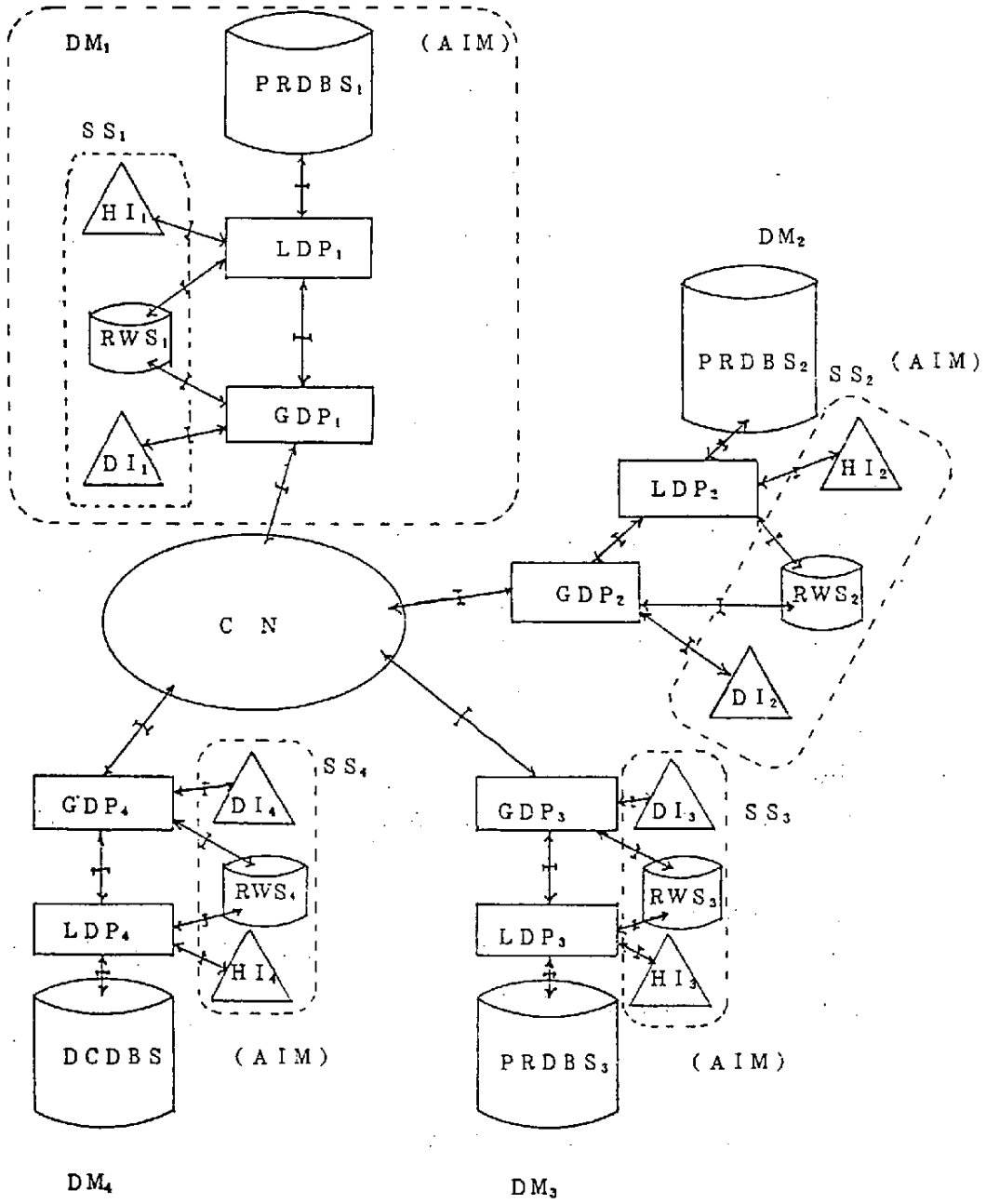


図 2.25　JDDBSの構成

作する。通信網ＣＮ及び，各プロセス間の通信は，ＡＩＭのＤＢ／ＤＣを用いて，放送通信をシュミレートしたものを設定している。

2.5.2 ＬＩＳ

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の構成である。

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の実際の構成については図3.4を参照のこと。

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，Sun 2／120 には第4章で述べるGranada II（関係型）がある。詳しくは，3.3を参照されたい。

2.5.3 ＬＩＰ

ＬＩＰの実現構成については5.2.6を参照されたい。

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間で通信処理を行う為のプロトコル，ＬＩＳ通信処理プロトコルについて述べる。

ＬＩＳ通信処理プロトコルの目的は，分散型データベースシステム（DDBS）の処理を効率的に行う事と，かつ設計を容易にする事である。この目的を実現する為に，群通信と呼ぶ機能を提供する。群通信は，複数のプロセス間での信頼性と順序性が確保された交互監視方式の非同期なデータ通信をサービスする。

以下に，群通信の概念と通信処理プロトコルの実現について述べるが，概念については昨年度の報告書で詳細に述べてあるので要点にとどめ，通信処理プロトコルの実現の記述に重点をおいて述べる。

3.1 ＬＩＳ通信モデル

本節では，ＬＩＳの通信処理階層と各階層の機能，およびＬＩＳの通信処理の特徴である群通信について述べる。

3.1.1 通信処理階層の構成及び各階層の機能

ＬＩＳの通信処理階層を図3.1に示す。

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DDBSの管理を行う。分散トランザクション管理（同時実行制御，コミットメント制御，信頼性制御，分散問合せ処理等）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。

(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。

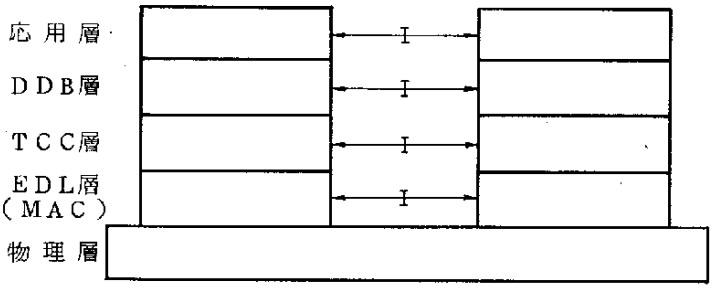


図3.1　ＬＩＳの階層構成

3.1.2　群　通　信

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を用いて行なわれる。以下，群通信の定義について述べる。

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）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（$i=1,\cdots,n$）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。

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

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

$\alpha \leq \beta$　*iff*　$\alpha < \beta$　$\alpha \equiv \beta$。

〔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.〔信頼一対一通信〕

$\forall S_i(m)$ 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。

O 2.〔FIFO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

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

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

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

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，…，Enの群（cluster）C（E_1，…，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 $\exists / S_i(m)$,

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

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。

C2〔FIFO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〔信頼放送性〕

C2〔FIFO性〕

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, ℓ）

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

3.2 トランスポート群制御プロトコル（TCCP）

TCCPとは，DDB層の実体に単純群サービスを提供する為に，トランスポート群制御（TCC）層の実体（TCCE）を共同動作（cooperate）させる為のプロトコルである。

以下にTCCPの概要について簡単に述べる。

3.2.1 群通信の機能と群の操作

A. 群通信の機能

LIS通信処理において，複数の単純群通信を行う為に必要な基本的機

能として，次の５つが考えられる。

- ○信頼放送性
- ○順序性
- ○単一放送路性
- ○フロー制御
- ○多重性

B. 群の操作

群の操作としては次の５つをサポートする。

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

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)，(iii)を合わせて，通信中という。

3.2.2 群の開設

群の開設の目的は，ＤＤＢ層の実体（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とる事にある。この手続きを行う期間を群の開設期という。

群の開設要求は，次の様に分られる。

- 群の開設要求
  - Active Open
  - Passive Open
    - 記　名　Passive Open
    - 無記名　Passive Open

Active Open（能動的開設）の開設要求を出すDDBEは能動的実体であり，Passive Open（受動的開設）を出すＤＤＢＥは受動的実体である。Passive

Openには群を構成する実体の集合を明記する記名Passive Openと，集合を明記しない無記名Passive Openとがある。Active Openは必らず集合を明記しなければならない。

群を構成する実体の集合は，群識別子で区別される。群識別子は以下の様に分られる。

$$\text{群識別子}\begin{cases}\text{ＯＣＩＤ（開設期群識別子）}=\{(S_1,\ell_1),\cdots,(S_n,\ell_w)\}\\ \text{ＣＩＤ（データ転送，解除期群識別子）}=(S_w,\ell_w)\end{cases}$$

ここで，S_iはＴＣＣ－ＳＡＰ（Service Access Point）の番地（ソケット番地ともいう）であり，ℓ_iはＳＡＰ内のＣＥＰ（Connection End Point）識別子である。また，ＣＩＤはＯＣＩＤの要素（S_i，ℓ_i）の中の最小値である。

群の開設とは，群内の全実体が共通のＯＣＩＤを認識しＣＩＤを得る事である。

共通のＯＣＩＤの認識とは，以下の条件を満足する事である。

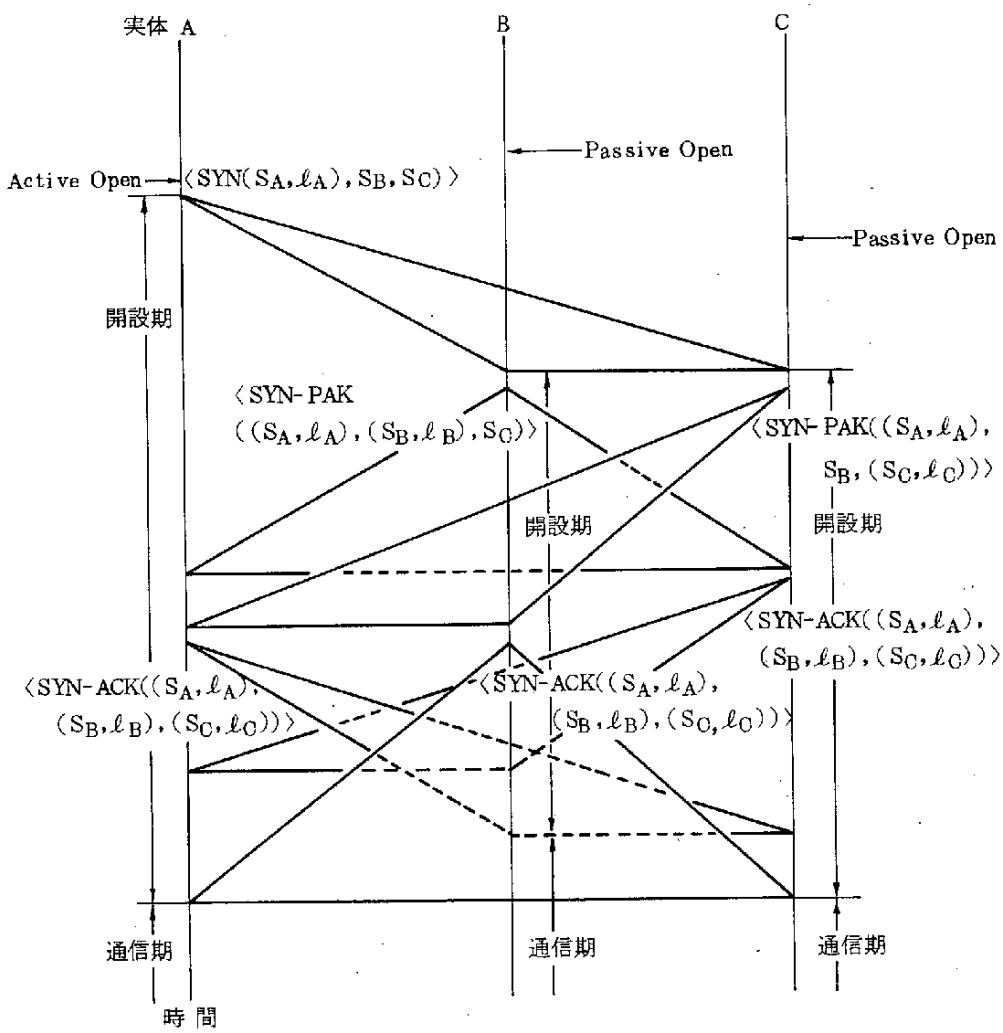
群内の実体を$E_1,\cdots,E_n$としたとき

① 各E_iが他の全E_jの（S_j，ℓ_j）を知っている。

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。

③ 各E_iは他のE_jが①，②の状態である事を知っている。

図3.2に3実体の群開設例を示す。



〈　　〉：セグメント（ＴＣＣ層のプロトコルデータ単位）を表わす。

図 3.2　3実体の群開設例

3.2.3　データ転送

群の開設で同期をとったＤＤＢＥは，全実体間で非同期にデータ転送を行う事が出来る。ＴＣＣＰはデータ転送の際，信頼放送性，順序性，単一放送路性を保証する。

A．単一放送路性

この機能はＥＤＬ層にＬＡＮを用いる事により保証出来る。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により，単一放送路性が確保される。

B．信頼放送性と順序性

信頼放送性と順序性は，ＴＣＣ層のプロトコルデータ単位（ＰＤＵ：Protocol data unit）であるセグメントに対して，通番の処理と２相Acknowledgement（２相確認）を行り事により保証される。通番の処理については，実現レベルの所で述べてある。

２相Acknowledgementは，以下の条件を満足する手順である。

群内の実体（ＴＣＣＥ）をE_1，…，E_i，…，E_nとしたとき，

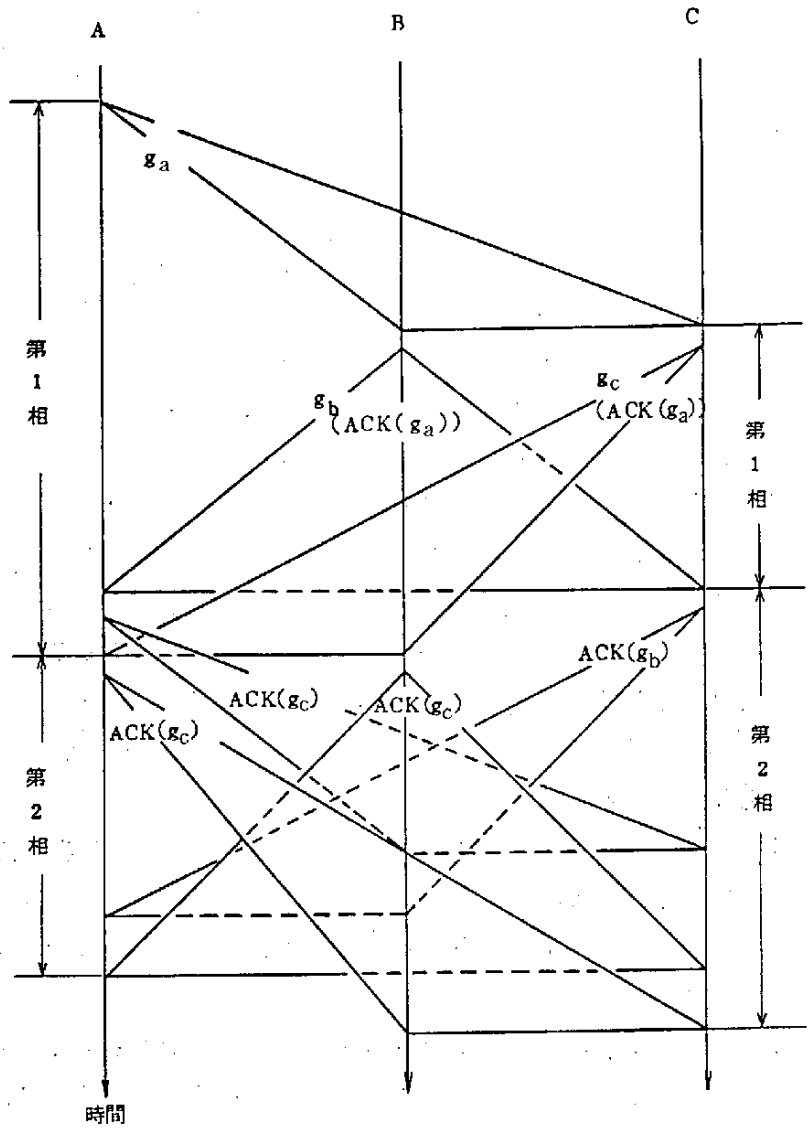
① 送信実体E_iが送信したセグメントg_iは他の全実体E_jにより正しく受信される。

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。

③ 送信実体E_i以外の全実体E_jがg_j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，群開設期のＯＣＩＤの認識に関する条件と同等である。

２相Acknowledgementは，図3.3の様に第１相と第２相とに分られる。第１相は上記の条件の①と②にあたり，第２相が条件③にあたる。また，セグメントg_iが条件①，②を満たす事をPreackされたといい，条件③を満たす事をAckされたという。



〈g_a〉：
〈g_b〉：　データセグメント
〈g_c〉：

〈ACK(n)〉：データセグメントnに対するACKセグメント

図 3.3　2相Acknowledgement 手順

3.2.4　群の解除

群の解除は，データ転送と同じ手順により行なわれる。全実体からのＦＩＮセグメントに対して，２相Acknowledgement がとれた時点で群が解除される。

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実体内のテーブル等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，加入と同様にテーブル等の変更を必要とする。

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　ＬＩＳ通信処理プロトコルの実現

本節では，ＬＩＳ通信処理プロトコルの実現レベルについて述べる。

3.3.1　開発環境と基本構成

ＬＩＳの開発に用いた機器の構成を図3.4に示す。

- ＬＡＮ

  図における同軸及びＮＩＵ部分

  アンガマンバス社のNet／One

  Ethernetに準拠した，ベースバンドの同軸ＬＡＮ

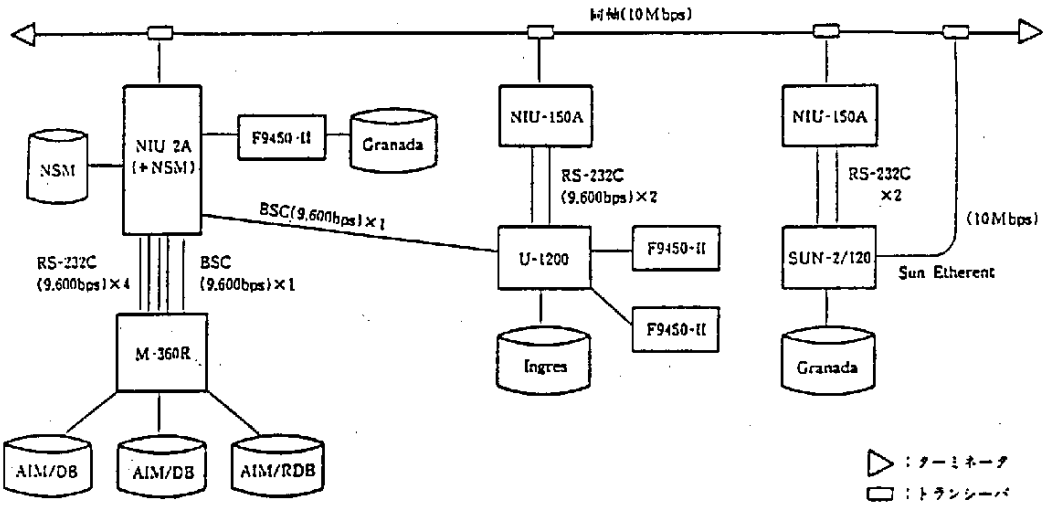


図3.4　ＬＩＳの構成

◦ＳＵＮ－２／１２０

ＳＵＮマイクロシステム社のエンジニアリングワークステーション。

ＯＳは，ＵＮＩＸ4.2ＢＳＤ。メモリー２ＭＢ，ディスク１６８ＭＢ。

◦Ｕ－１２００

パナファコムのミニコンピュータ。

ＯＳは，ＵＮＩＸ・SystemⅢ。メモリー１ＭＢ，ディスク１３０ＭＢ。

◦Ｍ－３６０Ｒ

富士通の大型コンピュータ。

ＯＳはＦ4／ＭＳＰ。メモリー１２ＭＢ。

ＬＩＳ通信処理プロトコルは，汎用性を考慮して，ＵＮＩＸのＣ言語を用いた。よって現状の機器構成では，Ｕ－１２００とＳＵＮ－２を対象としている。

ＬＩＳの通信階層とプロセスの関係を図3.5に示す。

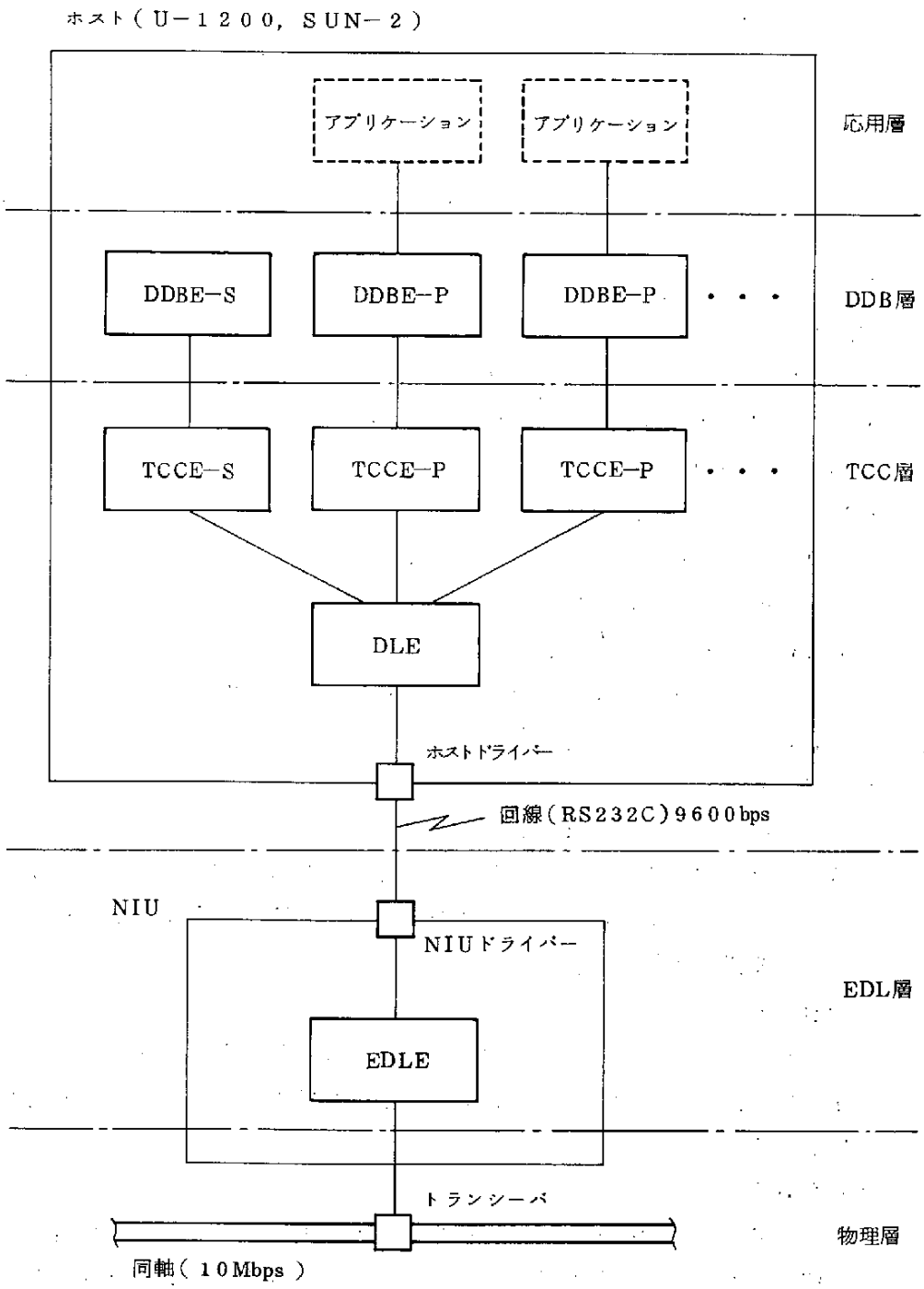


図 3.5　通信階層とプロセスの関係

図に示されている様に，ＥＤＬ層の実現はNet／Oneが備えているＥＤＬサービスをそのまま用いている。ＴＣＣ層は，ＤＬＥ，TCCE－S，TCCE－Pの３種類のプロセスに分けられる。ＤＤＢ層は，DDBE－S，DDPE－Pの２種類のプロセスに分けられる。

ＬＩＳ通信処理プロトコルという立場から見れば，ＤＤＢ層と応用層は，ＴＣＣ層以下の階層で提供する群通信を利用するアプリケーションである。今回の目的は，群通信の実現とその評価にあるので，ＤＤＢ層と応用層の区別はなく，ＤＤＢＥ－Ｐが両者を兼ねたプロセスである。

よって，実際に開発したのはＤＬＥ，TCCE－S，TCCE－P，DDBE－S，およびDDBE－Pの５種類のプロセスである。以下にその5種類のプロセスの役割について簡単に述べる。詳細については3.3.5～3.3.9を参照されたい。

◦ＤＬＥ

DLEの役割は，トランスポート群制御プロトコル（ＴＣＣＰ）の機能の中の多重化の実現と，Net／Oneとのインタフェースをとる事にある。

◦ＴＣＣＥ－Ｓ

ＴＣＣＥ－Ｓの役割は，ＤＤＢＥ－Ｐ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事である。

◦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事である。ＴＣＣＰの機能のうち，ＤＬＥがサポートする多重化以外の，信頼放送性，順序性，単一放送路性，フロー制御を実現する。

◦ＤＤＢＥ－Ｓ

ＤＤＢＥ－Ｓの役割は，利用者からの要求に基づきＤＤＢＥ－Ｐを起動する事と，ＬＩＳ通信処理プロトコル全体を管理する為の窓口を利用者に提供する事にある。

◦ＤＤＢＥ－Ｐ

ＤＤＢＥ－Ｐの役割は，ＴＣＣＥ－Ｐが提供する群サービスを用いて，上位層のユーザーに分散型データベースシステムのサービスを行う事にある。

3.3.2　ＵＮＩＸの機能

ＬＩＳ通信処理プロトコルは，Ｃ言語で開発され，ＵＮＩＸの下で動く様になっている。そして，プロトコルの機能を実現する為に，ＵＮＩＸのカーネルが備えている数々の機能を利用している。プログラムからこれらの機能を使用するには，システムコールと呼ばれる関数をコールすればよい。

ここでは，利用した機能の中でプロトコル実現の為に重要な役割をはたした，割込み処理機能，多重化処理機能，プロセス間通信機能，非同期入出力機能について述べる。

A．割込み処理機能

割込み処理は，signal 関数を用いる。

signal（シグナル番号，割込み処理ルーチン名）;

ここで，シグナル番号はシステムで定義されており，例えば，実行中のユーザープログラムに対して^Ｃ（コントロール＋Ｃ）で割込みをかけるシグナル番号は２となる。この関数はシグナル番号の割込みが発生した時に，割込み処理ルーチンに制御を飛ばす様にシステムに依頼するものである。

ＬＩＳ通信処理プロトコルでは，この割込み処理機能を，タイマー処理と異常等の割込みに対するバックアップ処理に用いている。図 3.6 にタイマー処理のしくみを示す。

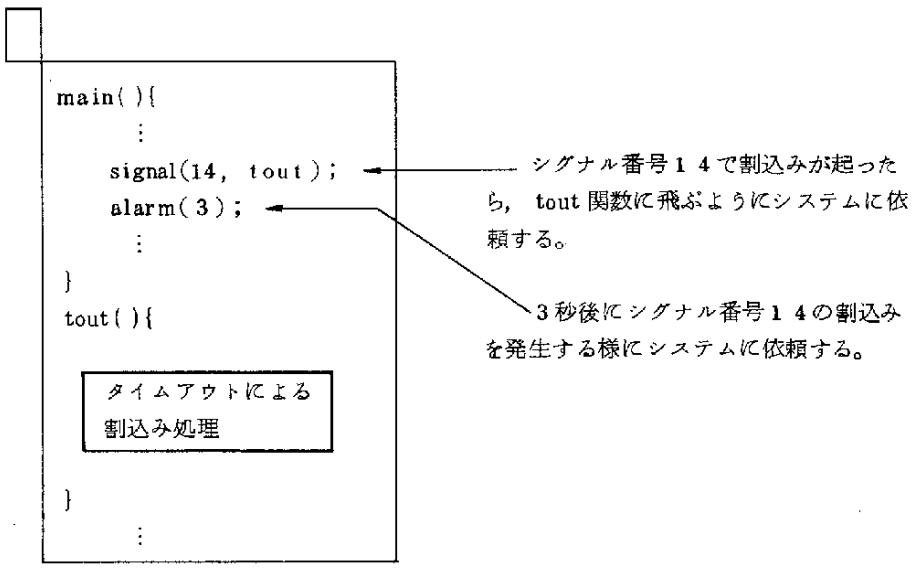


図 3.6　割込み処理機能を用いたタイマー処理のしくみ

B. 多重化処理機能

多重化処理は，fork関数とexec関数を用いて行なえる。

fork関数は次の様に用いる。

```
pid = fork();
```

fork関数を実行すると，そのプロセスのそこまでの実行環境をそのまま引き継いだ，まったく同じプロセスが新たに発生される。もとのプロセスは親プロセスと呼ばれ，新たに発生されたプロセスは子プロセスと呼ばれる。実行環境はそのまま引き継がれるので，親プロセスがforkされた時点で使用出来るファイルは，子プロセスでも使用可能である。fork関数のリターン値pidは，親プロセスには子プロセスのプロセスＩＤ（ＵＮＩＸがシステム内のプロセスを一意に識別するＩＤ）が返り，子プロセスには0が返る。よって，この値により，その後の親プロセスと子プロセス，独自の処理を記述する事が出来る。

fork関数による実行の流れを図3.7に示す。

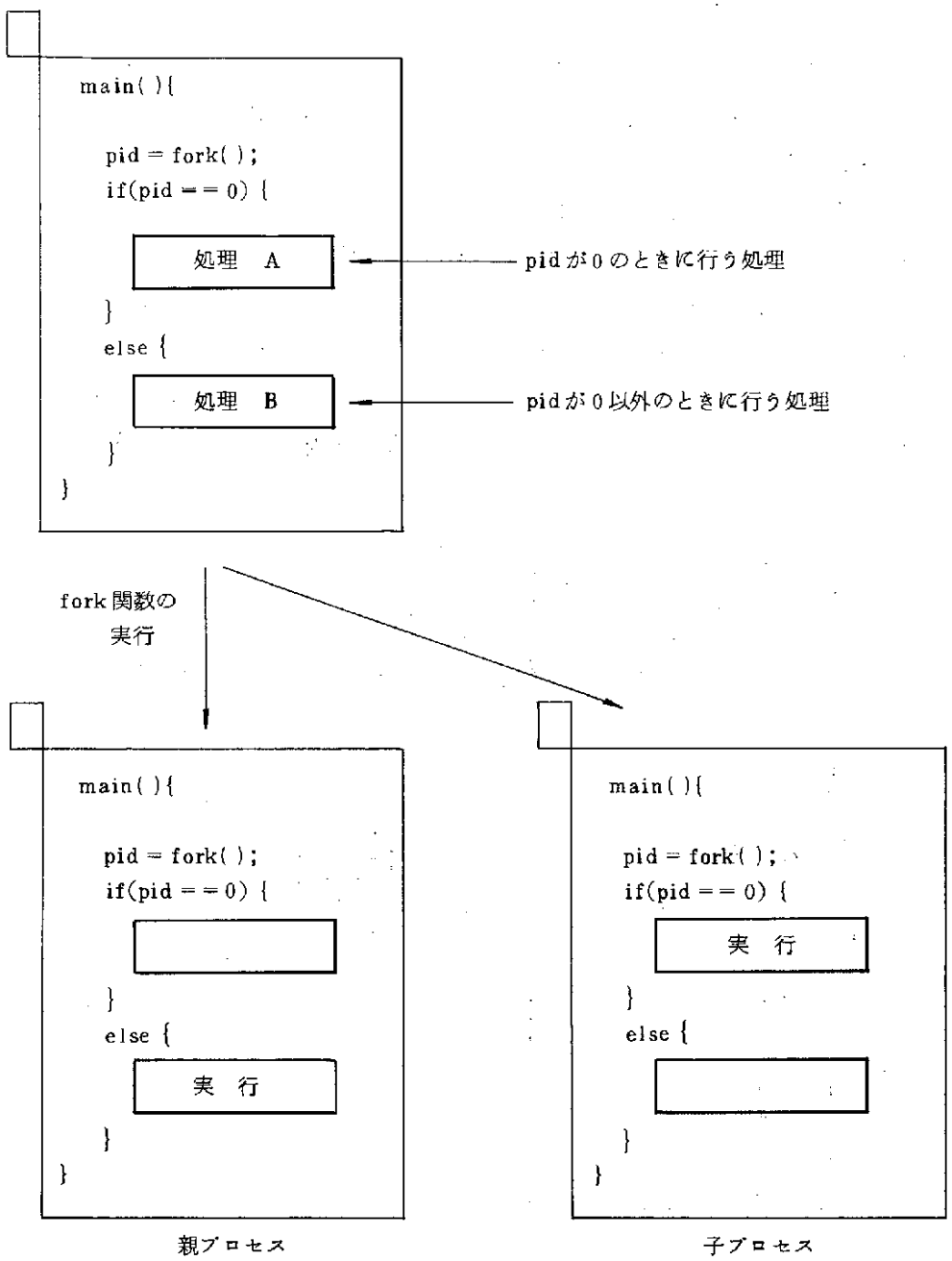


図 3.7　fork関数の実行の流れ

一方の exec 関数は，呼び出しプロセスを新しいプロセスで置き換える。exec 関数は実際には引数の記述のし方の違いから，いくつかのタイプのシステムコールが用意されているが，ほぼ次の様な記述で用いる。

exec（新しいプロセスのファイル名，新しいプロセスに渡す引数の列）;

exec 関数による実行の流れを，図 3.8 に示す。

以上の，fork 関数と exec 関数を組み合わせる事により，Aというプロセスから，Bというプロセス，Cというプロセス，…という多重化の処理が行なえる。ちなみに，ＵＮＩＸシステムのマルチ・ユーザー，マルチ・タスク機能も全て，この fork 関数と exec 関数を用いて実現されている。図 3.9 にAプロセスからBプロセスを起動する場合の実行の流れを示す。

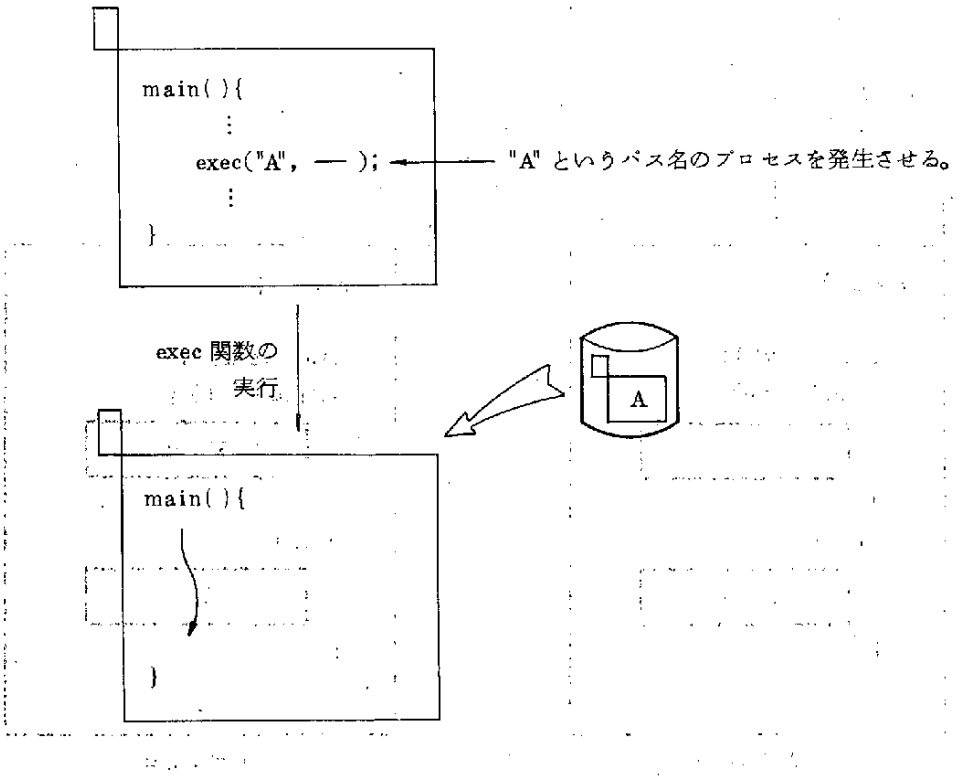


図 3.8　exec 関数の実行の流れ

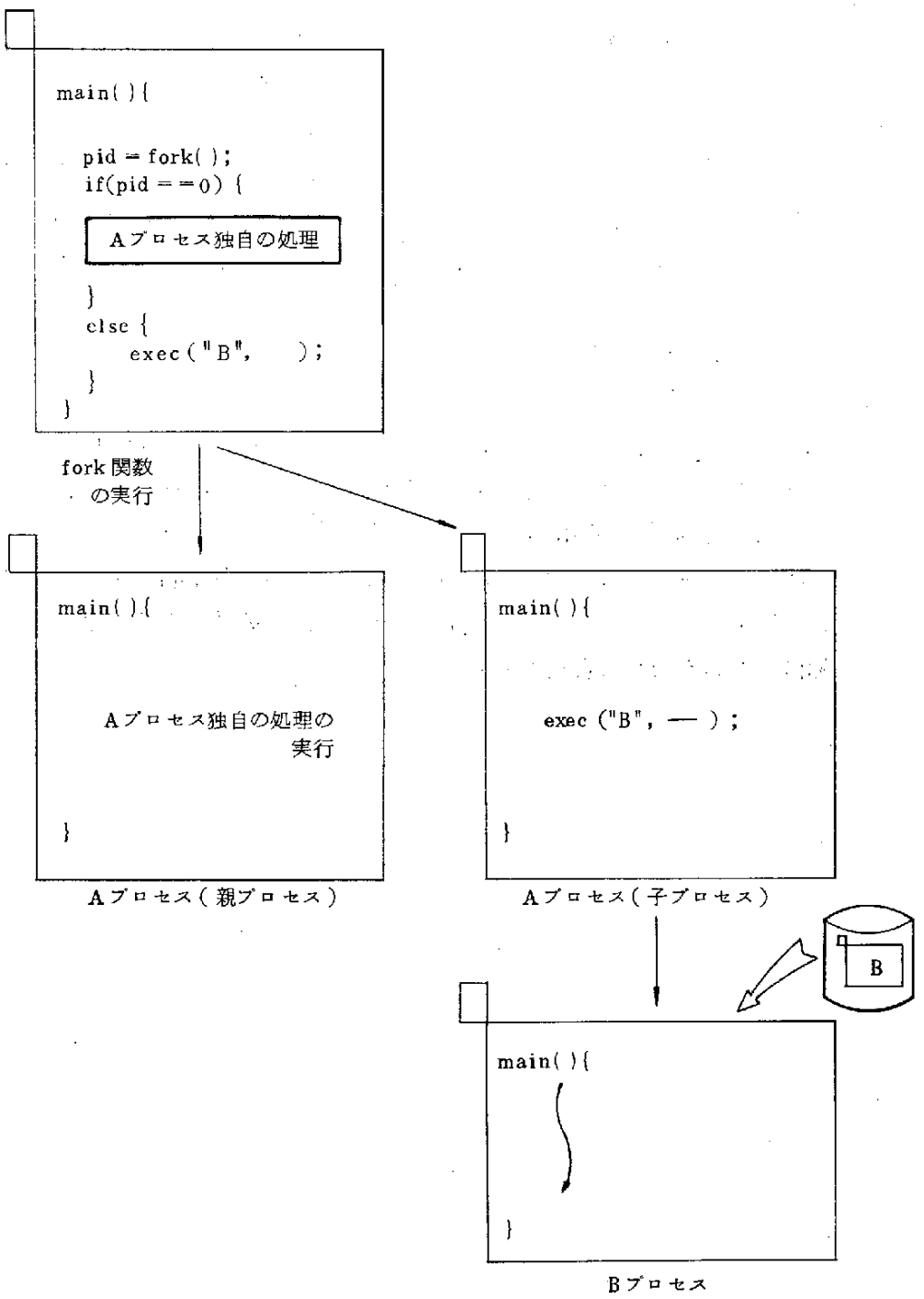


図 3.9　fork，exec関数を用いた多重化処理の流れ

C. プロセス間通信機能

ＵＮＩＸが備えているプロセス間通信機能はSystemⅢ（Ｕ－１２００上）と4.2ＢＳＤ（ＳＵＮ－2上）とでは大きく異なる。

SystemⅢでは，パイプと呼ばれる機能（pipe関数で実現される)を持っている。パイプは，親プロセスがfork関数を用いて生成した子プロセスとの間にのみ設定出来る。パイプによる通信は，一般のファイルに対するread，writeと同等で，ファイル管理テーブル上にファイル記述子が設定される。

pipe関数は次の様に用いる。

```
int fd〔2〕;      pipe(fd);
```

pipe関数を実行すると，２つのファイル記述子が返る。fd〔0〕は読み取りモードでオープンされ，fd〔1〕は書き込みモードでオープンされる。図3.１０にpipe関数の使用例を示す。

図では，親プロセス側がwrite，子プロセス側がreadとなっているが，これは逆でもかまわない。ただし，１回のpipe関数の実行では片方向の通信しか行えない点に注意がいる。

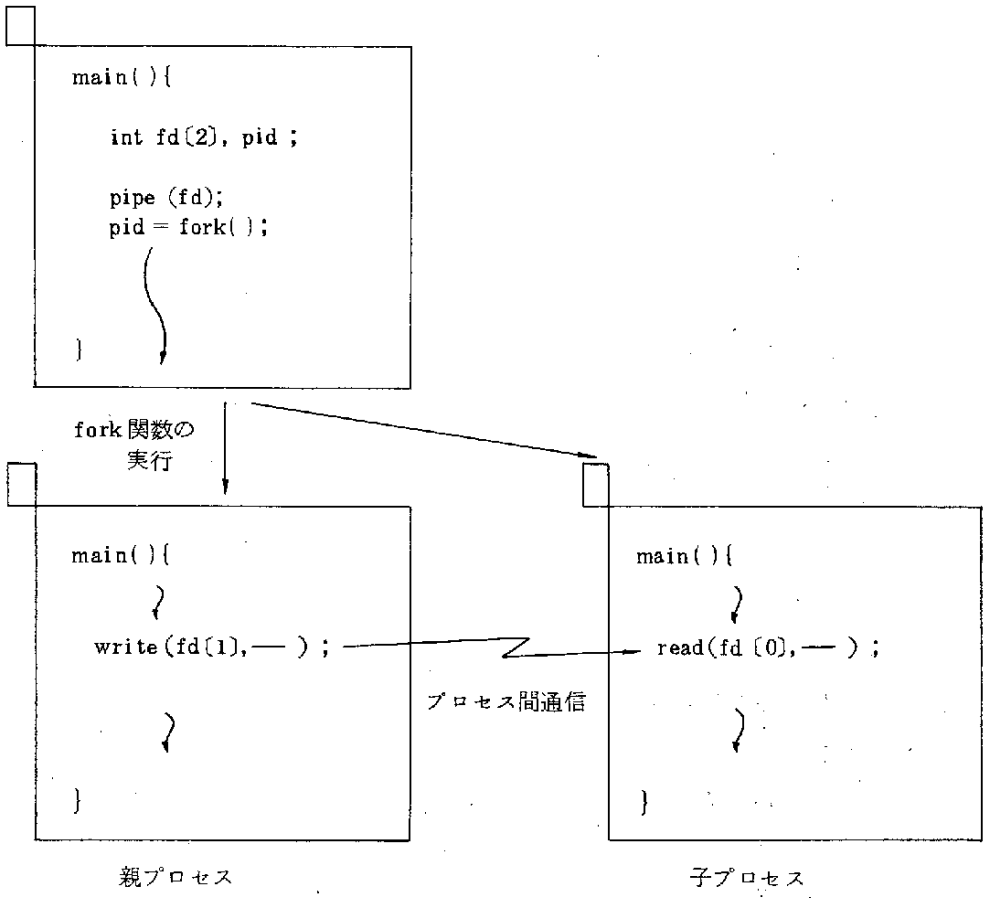


図 3.1 0　pipe関数の使用例

4.2 BSDのUNIXでは，OSの機能としてDARPAのTCP／IPプロトコルが組み込まれている。従って，SUN－2ではパイプ機能の様に同一ホスト内でのプロセス間通信だけではなく，Ethernet を介して他ホストのプロセスとの通信も簡単に行なえる。System Ⅲのパイプ機能との大きな違いは，パイプ機能は必ず fork 関数を使う必要があるという事，すなわち，元（根）は一つのプロセスでなければならないのに対し，TCP／IP プロトコル用の関数では通信を行うプロセスをそれぞれ別個に起動出来る点である。なぜならば，TCP／IPでは，相手のプロセスのアドレスを指定して，そのプロセスとコネクションをはるからである。ただし，UNIX相互の互換性の問題から，4.2 BSDでもシステム・サブルーチ

ンとしてpipe関数をサポートしている。

D. 非同期入出力機能

非同期入出力機能は，fcntl関数を用いて行なえる。

fcntl関数はオープンされているファイル記述子fdに対して，次の様に用いる。

fcntl（fd，F_SETFL，O_NDELAY）；

ここで，F_SETFL，O_NDELAYはシステムのインクルードファイル＜fcntl.h＞に定義されている値である。また，ファイル記述子fdはpipe関数で得たファイル記述子を用いる。

通常のパイプに対するreadは，データがパイプに存在すればread出来るが，データが無い場合には待たされる。しかし，上記のfcntl関数を実行したパイプに対する入力は，データが無い場合でもプロセスに直ちに制御が戻される。

3.3.3 ＬＩＳ通信処理プロトコルの起動方法と利用方法

A. 起動方法

ＬＩＳ通信処理プロトコルを起動するには，shell（ＵＮＩＸのコマンド・アナライザー）に対して次の様にコマンドを入力する。

% SEED PARAFア　（下線：入力部分）

ここでＳＥＥＤとは，プロトコルの一番もととなるプロセスのパス名である。ＳＥＥＤプロセス（プロセス名もＳＥＥＤと呼ぶ事にする）は，ＵＮＩＸでのinitプロセスとshellプロセスの関係と同様で，プロトコル内の全プロセスの一番上位の親プロセスとなる。引数のPARAFは，ＳＥＥＤプロセスが用いるパラメータ・ファイル名で，ＤＬＥ，ＴＣＣＥ－Ｓ，ＴＣＣＥ－Ｐ，およびＤＤＢＥ－Ｓの各プロセスのパス名と，各プロセスにインタフェース・ファイルを介して渡されるパラメータ値が含まれている。PARAFの内容については，3.3.4を参照の事。

ＳＥＥＤプロセスの目的は，ＤＬＥ，ＴＣＣＥ－Ｓ，およびＤＤＢＥ－Ｓの各プロセスを起動する事と，各プロセス間にパイプを用意する事である。ＳＥＥＤプロセスからＤＬＥ，ＴＣＣＥ－Ｓ，ＤＤＢＥ－Ｓの各プロセスが起動される手順を図3.11に，又各プロセス起動後の関係を図3.12に示す。

ＵＮＩＸ内の１プロセスが同時に保持出来るファイル記述子の数は２０までであるので，図3.１２の様にパイプを用意しようとすると，プロセス間で最大６となる。よって双方向の通信路を１つのパスと呼ぶ事によると，プロセス間で持てるパスの数は最大で３である。各パスは番号が付けられており，それぞれ，パス０，パス１，パス２と呼ばれる。パス０は管理用であり，パス１，２は通信用である。

B．利用方法

ＬＩＳ通信処理プロトコルが上記のＡの方法により起動されると，ＤＤＢＥ－Ｓからコンソールに対して次のプロンプトが表示される。

＊＊＊＊＊　DDBEP？

これに対し，利用者がＤＤＢＥ－Ｐのパス名を入力する事によりＤＤＢＥ－Ｐが起動される。起動されたＤＤＢＥ－Ｐが群開設要求を出し，それに対する準備が整うと，再び上記のプロンプトが表示される。利用者は，さらにＤＤＢＥ－Ｐを起動する場合にはパス名を，起動しない場合にはリターンキーのみを押下する。以下に，利用者がＤＤＢＥ－Ｐのパス名を入力してから，再びプロンプトが表示されるまでの手順を示す。

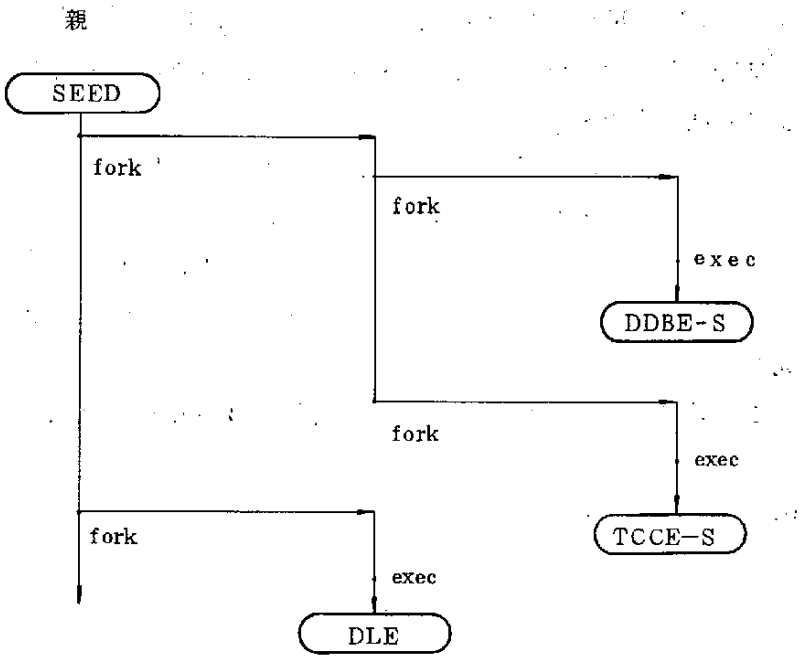


図3.１１　DLE，TCCE－S，DDBE－Ｓの起動手順

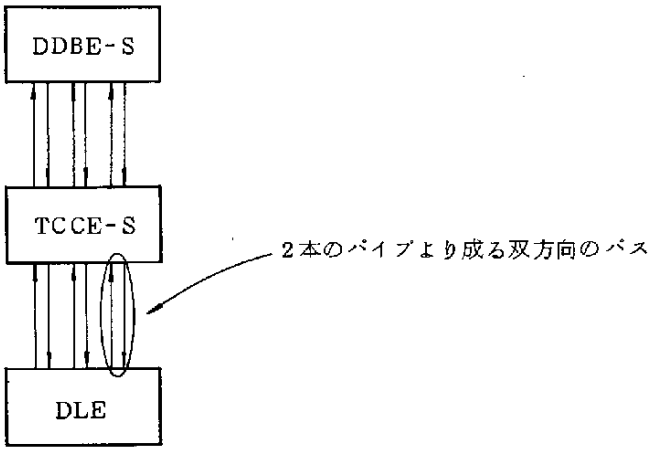


図 3.1 2　DLE，TCCE−S，DDBE−S

① ＤＤＢＥ－Ｓは fork し，さらに入力されたＤＤＢＥ－Ｐのパス名で exec する。

② 起動されたＤＤＢＥ－Ｐは，親プロセスであるＤＤＢＥ－Ｓのパスを受け継いでいるので，パス０を通じＴＣＣＥ－Ｓに対して群開設要求（C−AOPEN又はC−POPENフレーム）を出す。

③ 群開設要求を受け取ったＴＣＣＥ－Ｓは，さらにパス０を通じてＤＬＥに群開設要求を出す。

④ ＤＬＥは群開設要求を受けると，あいている通信用のパス i をさがし，パス０を通じてＴＣＣＥ－Ｓにパス i を通知する（Ｃ－ＡＣＫフレーム）。

⑤ ＴＣＣＥ－Ｓは fork し，さらに exec してＴＣＣＥ－Ｐを起動する。このときインターフェースファイルTPIFを介して，TCCE−Pにパス i を通知する。

⑥ ＴＣＣＥ－Ｓはさらに，パス０を通じてＤＤＢＥ－Ｐにパス i を通知する。

図 3.1 3に①～⑥の手順を示す。

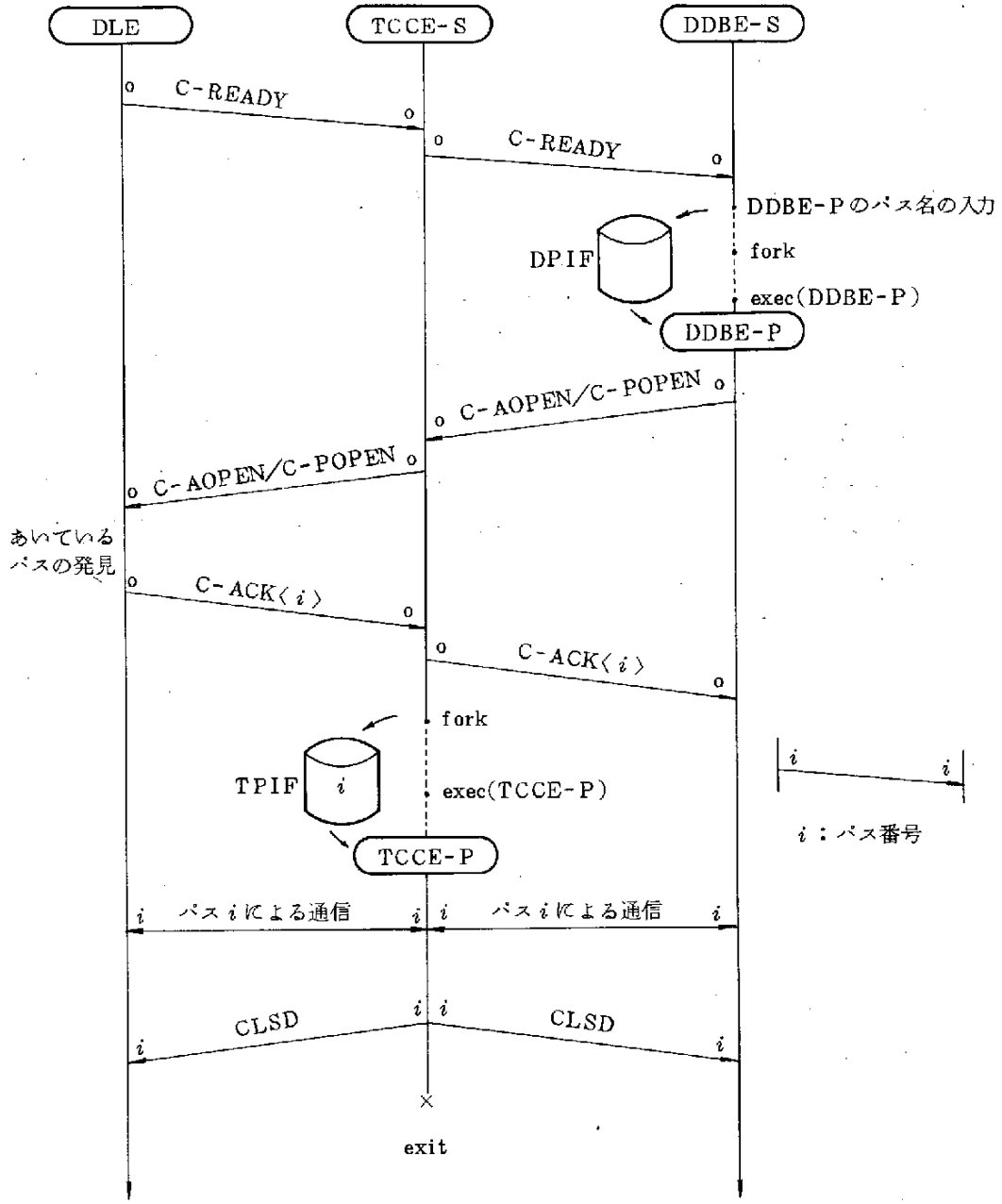


図 3.1 3　各プロセスの処理手順

3.3.4 プロセス間インターフェース

各プロセス間のインターフェースとしては，通信データと，インターフェース・ファイルの2種類がある。

A．通信データ

通信データは，ＤＤＢ層，ＴＣＣ層，ＥＤＬ層，および物理層内の各プロセス間又は媒体上で転送されるデータ単位である。ＤＤＢ層とＴＣＣ層内ではパイプを通して転送される。図3.14に各プロセス間の通信データ単位を示す。図において，ＴＣＣ－ＳＡＰ間を点線で転送されているストリームは，ＤＤＢ層のプロトコルデータ単位（ＰＤＵ）である事を示している。また，ＥＤＬ－ＳＡＰ間を点線で転送されているセグメントは，ＴＣＣ層のプロトコルデータ単位である事を示している。プロトコルデータ単位とは，同一プロトコル内のpeer実体間での論理的なデータ転送単位である。

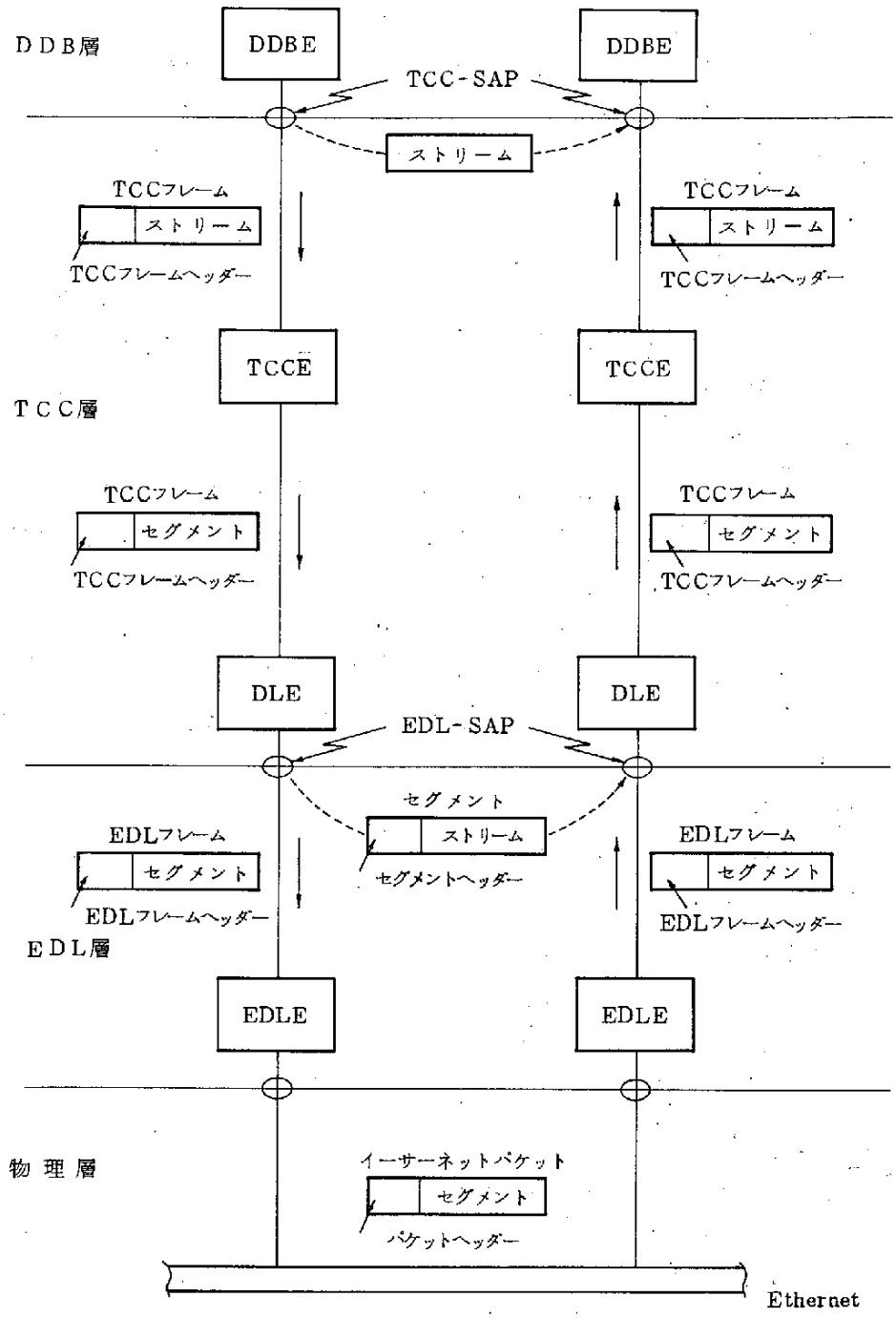


図 3.1 4　通信データ単位

通信データ単位は一般に，管理用のものと通信用のものとに分かれる。図3.14は通信用のものである。管理用の通信データは隣接するプロセス間でのみ意味を持つ，コントロール用のデータ単位である。

以下に各通信データ単位のフォーマットについて述べる。

(i) ストリーム

ストリームはDDB層のプロトコルデータ単位の呼び名である。ストリームはストリームヘッダーとストリームデータとに分けられる。フォーマットの詳細については3.3.9で述べる。ストリームは基本的には，任意長のバイト列のデータである。

(ii) TCCフレーム

TCCフレームは，TCC層のインターフェースデータ単位の呼び名である。TCCフレームはTCCフレームヘッダーとTCCフレームデータとに分けられる。図3.15にフォーマットを示す。

表3.1にTCCフレームヘッダー内のコマンドにより識別されるTCCフレームの一覧を示す。コマンドがDATAの場合にTCCフレームデータ部にストリーム又はセグメントが格納される。またコマンドの種類によってはTCCフレームデータのないものもある。

(iii) セグメント

セグメントはTCC層のプロトコルデータ単位の呼び名である。セグメントはセグメントヘッダーとセグメントデータとに分けられる。図3.17にフォーマットを示す。

セグメントヘッダーの各フィールドの意味は次のとおりである。

- 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3.3参照〕である。

◦ 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3.3参照〕。

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（2 byte）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2 byte boun-

dary 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3.１８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。

(iv) ＥＤＬフレーム

ＥＤＬフレームは，ＥＤＬ層のインターフェースデータ単位（ＩＤＵ）の呼び名である。ＥＤＬフレームはＥＤＬフレームヘッダーとＥＤＬフレームデータとに分けられる。図3.１６にフォーマットを示す。ＥＤＬの機能（FUNCTION CODE）によってはＥＤＬフレームヘッダーが異なり，ＥＤＬフレームデータがないものがある。表3.２にＥＤＬフレームの一覧表を示す。

(v) パケット

ＥＤＬ層のプロトコルデータ単位をパケットと呼ぶ。パケットのフォーマットはEthernetパケットと等しい。パケットのフォーマットを図3.１９に示す。

ＬＩＳ通信処理プロトコル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。あるＥＤＬＥ（ＥＤＬ実体）から送出されたパケットは網内の全ＥＤＬＥで受信される。

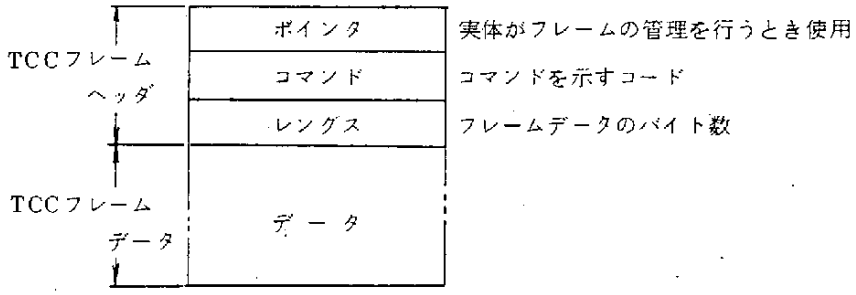


図3.15　TCCフレームフォーマット

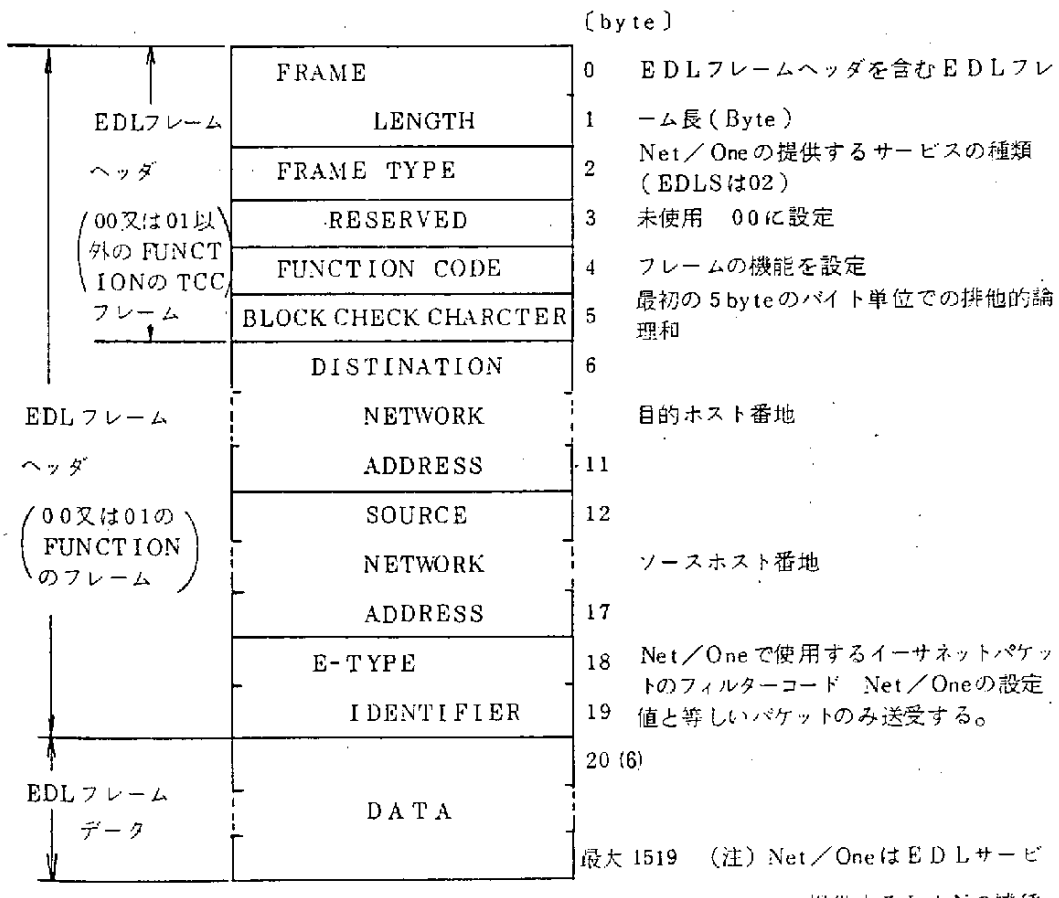


図3.16　EDLフレームフォーマット

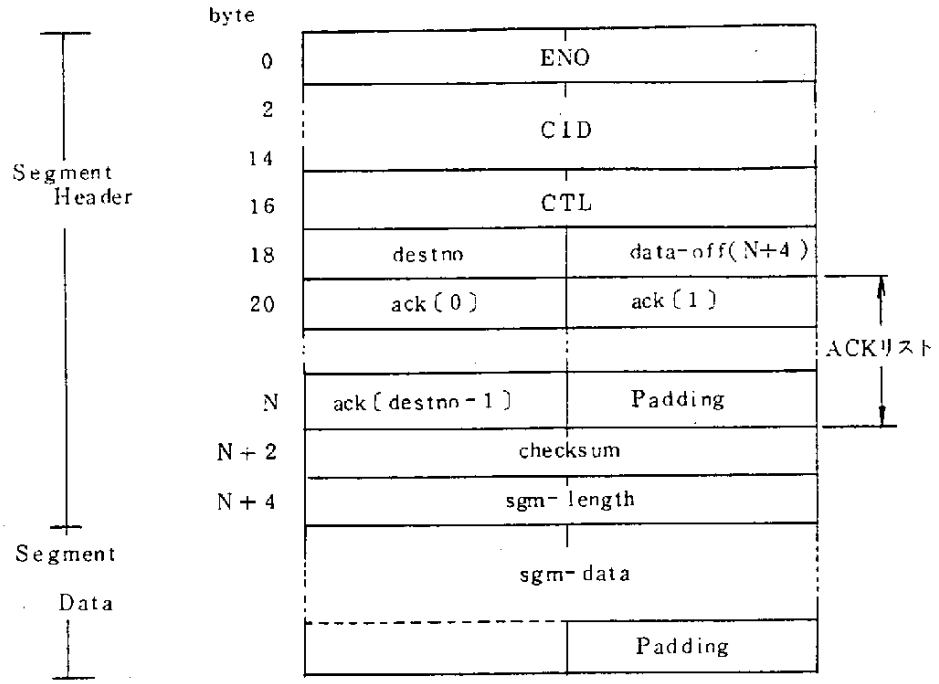


図 3.1 7　セグメントのフォーマット

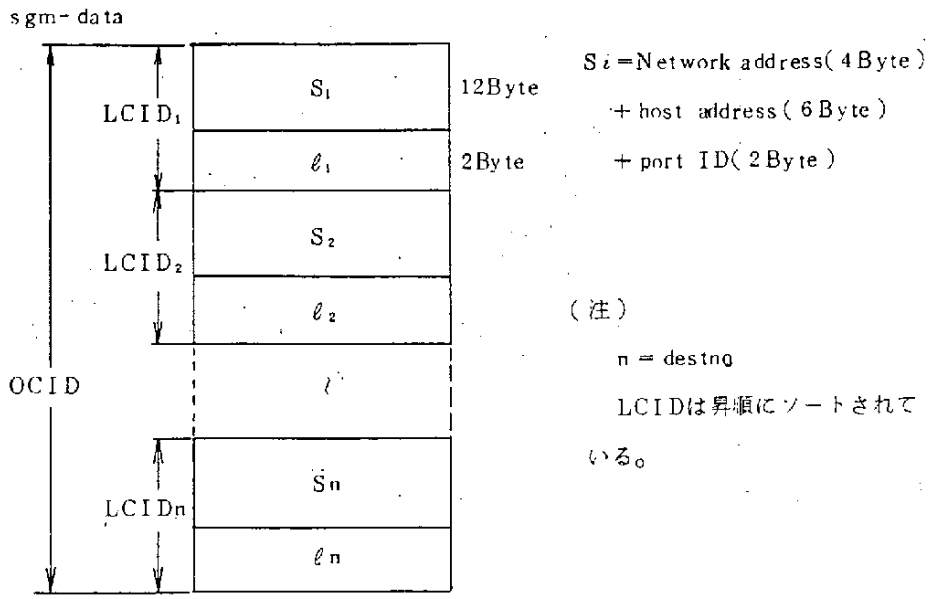


図 3.1 8　群開設時のセグメントデータ

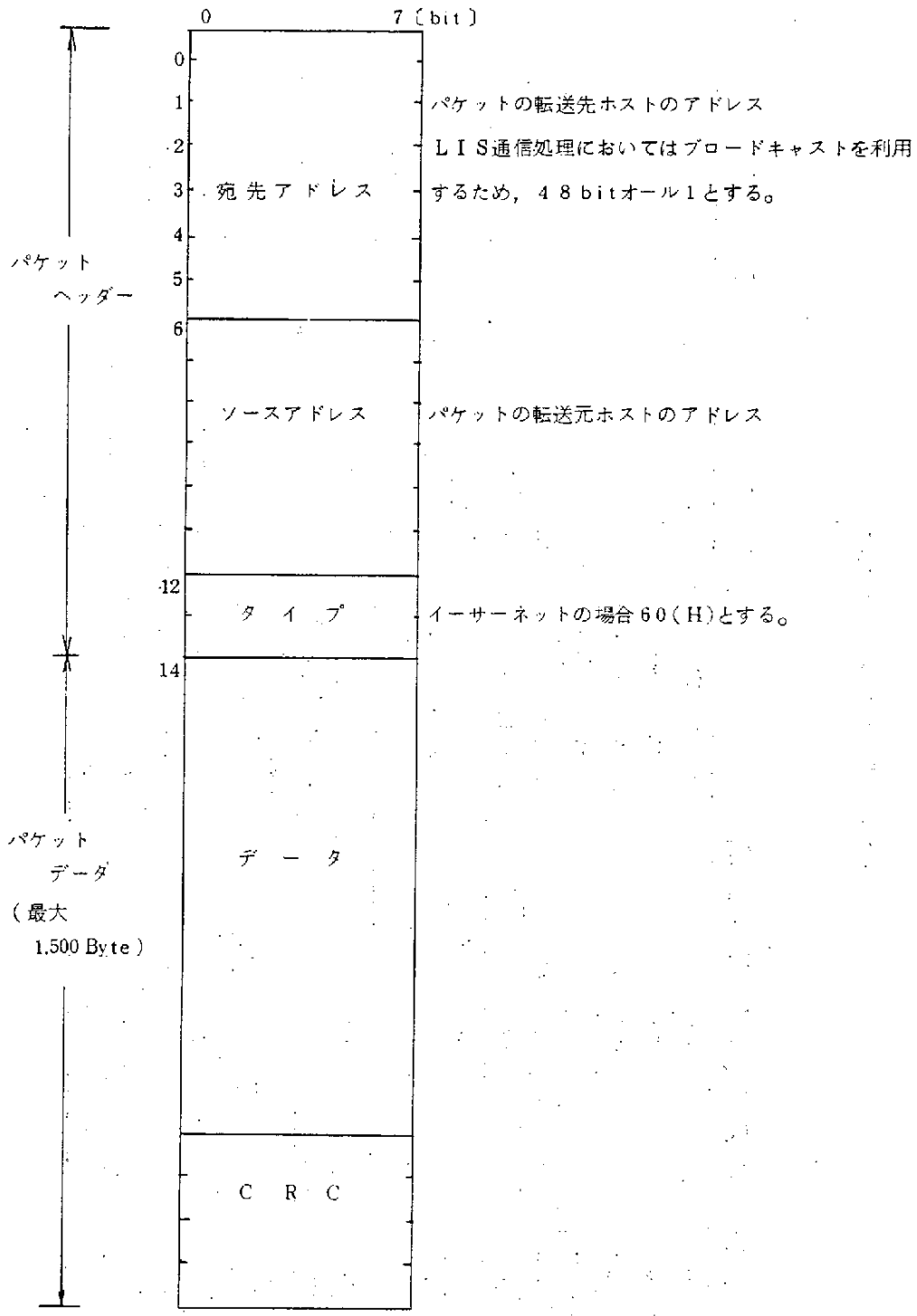


図3.19　イーサーネットパケットフォーマット

表3.1　ＴＣＣフレーム一覧表

コマンド	コード	DDBE--TCCB	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。

表3.2　ＥＤＬフレーム一覧表

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6 Byte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

表 3.3　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 MODE	Data Field 使用有無	Data Field 内容	ACK Field 使用有無	ACK Field 内容	応答使用の有無	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>					○					○							×		×		×	
	REJECT	<REJ>								○						△			×		○	Smallest	×	
		<REJ-PAK>								○	○					△			×		○	ACK Table	×	
		<REJ-ACK>								○		○							×		○		×	
	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	×	
	データ転送	<DATA>													○	△	△	△	○	Data	○	ACK	○	POLLとFINAL は排他使用
		<ACK>										○					△		×		○	ACK	○	

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。

B. インターフェースファイル

インターフェースファイルはホスト内の各プロセス（ＳＥＥＤ，ＤＬＥ，ＴＣＣＥ－Ｓ，ＴＣＣＥ－Ｐ，ＤＤＢＥ－Ｓ，ＤＤＢＥ－Ｐ）が起動された時に，利用者又は他のプロセスから渡されるファイルで，各プロセスの設定に必要なパラメータ値を含んでいる。

インターフェースファイルには次の６種類がある。

- ＰＡＲＡＦ：ＳＥＥＤプロセス用。利用者からアーギュメントで渡されるインターフェースファイル。
- ＤＬＩＦ：ＳＥＥＤからＤＬＥに渡されるインターフェースファイル。
- ＴＳＩＦ：ＳＥＥＤからＴＣＣＥ－Ｓに渡されるインターフェースファイル。
- ＤＳＩＦ：ＳＥＥＤからＤＤＢＥ－Ｓに渡されるインターフェースファイル。
- ＴＰＩＦ：ＴＣＣＥ－ＳからＴＣＣＥ－Ｐに渡されるインターフェースファイル。
- ＤＰＩＦ：ＤＤＢＥ－ＳからＤＤＢＥ－Ｐに渡されるインターフェースファイル。

ＤＬＩＦ，ＴＳＩＦ，ＤＳＩＦ，ＴＰＩＦ，およびＤＰＩＦについては，3.4.5項以降の各プロセスごとの記述の所で述べる。ここではPARAFインターフェイスファイルについて述べる。PARAFインターフェイスファイルにはＬＩＳ通信処理プロトコルの管理者が設定可能なパラメータ値を全てセット出来る。これらのパラメータ値は，他のインターフェースファイルを経由してそれぞれのプロセスに渡される。

(i) PARAFファイルのフォーマット

PARAFファイルはＵＮＩＸの端末属性ファイル“ termcap ”と同様，文字列でパラメータを指定する。

１カラム名

↓

```
# System file No 1…                              ← ①
DLIF:DLE  interface file…                        ← ②
        :TMV0=30:TMV1=30:BAUD=9600               ← ③
        :NIU="/dev/ttys1"                        ← ④
```

← ⑤

TSIF：TCES　interface file…　← ⑥

① 1カラム目が'#'で始まる行は，コメント行。

② インターフェースファイルを識別する。1カラム目から書かねばならない。次に表われるインターフェースファイルの識別までが，そのインターフェイスファイルに対するパラメータとなる（この場合，③④がDLIFのパラメータ）':'以降はコメントとなる。

③ パラメータ行，先頭にはタブか，又は1つ以上の空白が必要。又最初の文字は':'でなければならない。各パラメータは':'で区切る。各パラメータは以下の型をとる。

パラメータID＝パラメータ値
　　└→整数の場合は直接入力
　　文字　〃　　“　”で囲む。

④ パラメータ値が文字列なので“　”で囲まれている。

⑤ 空白行はスキップする。

⑥ 次のインターフェースファイル識別行。②を参照の事。

(ii) PARAFファイルのパラメータ値

各インターフェイスファイルのパラメータIDは各々のインターフェースファイルを読み辺むプロセスの変数名と等しい。

以下に各パラメータIDの意味を示す。

イ) SEEDパラメータファイル（SEEDプロセス自身が用いるパラメータ）

DLE　… DLEの実行形式のパス名
TCES　… TCES　〃
TCEP　… TCEP　〃
DDBES … DDBES　〃

ロ) DLIFインターフェイスファイル

TMV0　…ステート遷移にかかるタイマー値。遷移図のGTSの値。
TMV1　…　〃　〃　TSの値。

TOMAX…State＝NIU-Cleared-wait時のタイムアウトの再試行回数。

タイウアウト回数＜TOMAX→CLEAR-NIU再送

〃　　≧　　→C-ABORT再信後，終了

BAUD　…ＮＩＵ用のＴＴＹドライバーのボーレート。ただし，現在はUNIX4.2BSDのプロセス間通信機能を用いているので未使用。

ＮＩＵ　…ＮＩＵ用のＴＴＹドライバーの特殊ファイル名。

ハ) ＴＳＩＦインターフェースファイル

TMV0　…ステート遷移にかかるタイマー値，遷移図のGTSの値。

TMV1　…　〃　〃　ＴＳの値。

ただし，現在C-STATEは無視しているので未使用。

ニ) ＴＰＩＦインターフェースファイル

QTM1　…RTQ1キュー内のセグメントにかかるタイマー値。RTQ1にキューイングされる時，又は再送後のセグメントに対してセットされる。タイムアウト時の処置はＱＭＸ１参照。

QTM2　…ＲＴＱ２，ＡＱ２キュー内のセグメントにかかるタイマー値。両者にキューイングされるときにセットされる。

ＲＴＵ２内のセグメントのタイムアウト

→ＲＴＱ２よりはずす

ＡＱ２　〃

→AQ2よりSQキューへ

STM12
〜
STM66　…ステート遷移にかかるタイマー値。

QMX1　…ＲＴＱ１内のセグメントのタイムアウト時の最大再送回数。

タイムアウト回数＜QMX1→セグメントの再送

〃　≧　〃　→エラー

QMX2　…ＱＴＭ２に対するものであるが，意味を持たない。

未使用。

QMX12 ～ SMX66 … STM12～STM66に対応する各ステートのタイムアウト時のタイムアウト処理の最大履行回数。

タイムアウト回数＜SMX？？→ステートごとに異なる処理

〃 ≧ 〃 → エラー

AOMX … State＝Syn-sent時のタイムアウトの再試行回数。

タイムアウト回数＜AOMX→リトライ処理

〃 ≧ 〃 →アボート

RRMX … RNR状態→RR状態に移る為の条件値。

RNR後に解放したメモリースペースの合計＞RRMX→RRへ

SDMAX…セグメントデータの最大長（バイト）。

ACV …タイマークロック値（秒）"alarm"システムサブルーチンの引数となる。最小値＝1。

WS …ウィンドーサイズ。

ある実体からのセグメントの通番のぬけ＞WS→RST状態

〃 ≦ 〃 →REJ状態

3.3.5 DLE

A. 概 要

DLEはトランスポート群制御プロトコル（TCCP）の機能の内の多重化の実現と，EDL層のNet／OneのNIUとのインターフェースをとる働きをする。

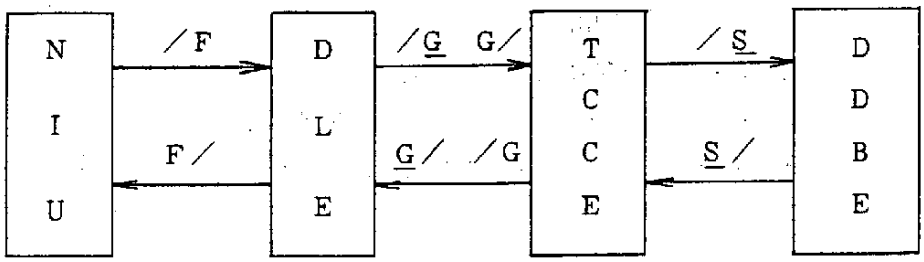
B. 状態遷移

状態遷移図の記法を図3.20に示す。

DLEの状態遷移は，Net／OneからのEDLフレームの受信による遷移〔図3.21～図3.23〕と，TCCE－P間とのパスiにおけるTCCフレームの受信による遷移〔図3.24，図3.25〕に分けられる。

C. インターフェースファイル

DLEはSEEDプロセスからDLIFインタフェースファイルを渡される。図3.26にフォーマットを示す。



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
{〈G 1〉 … 〈Gm〉}／：G 1，…，Gm のうちのいずれかのセグメントの受信
{〈G 1〉 … 〈Gm〉}//：G 1，…，Gm のうちいずれかのセグメントを全ＴＣＣＥから受信

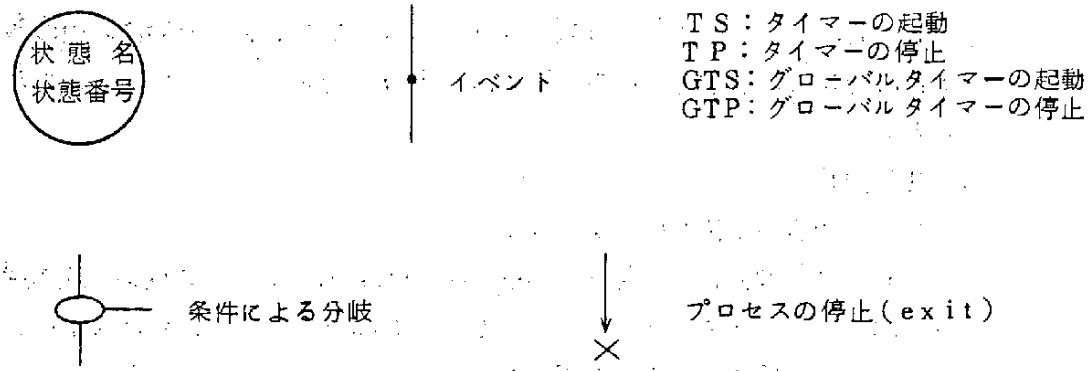


図 3.2 0　状態遷移図の記法

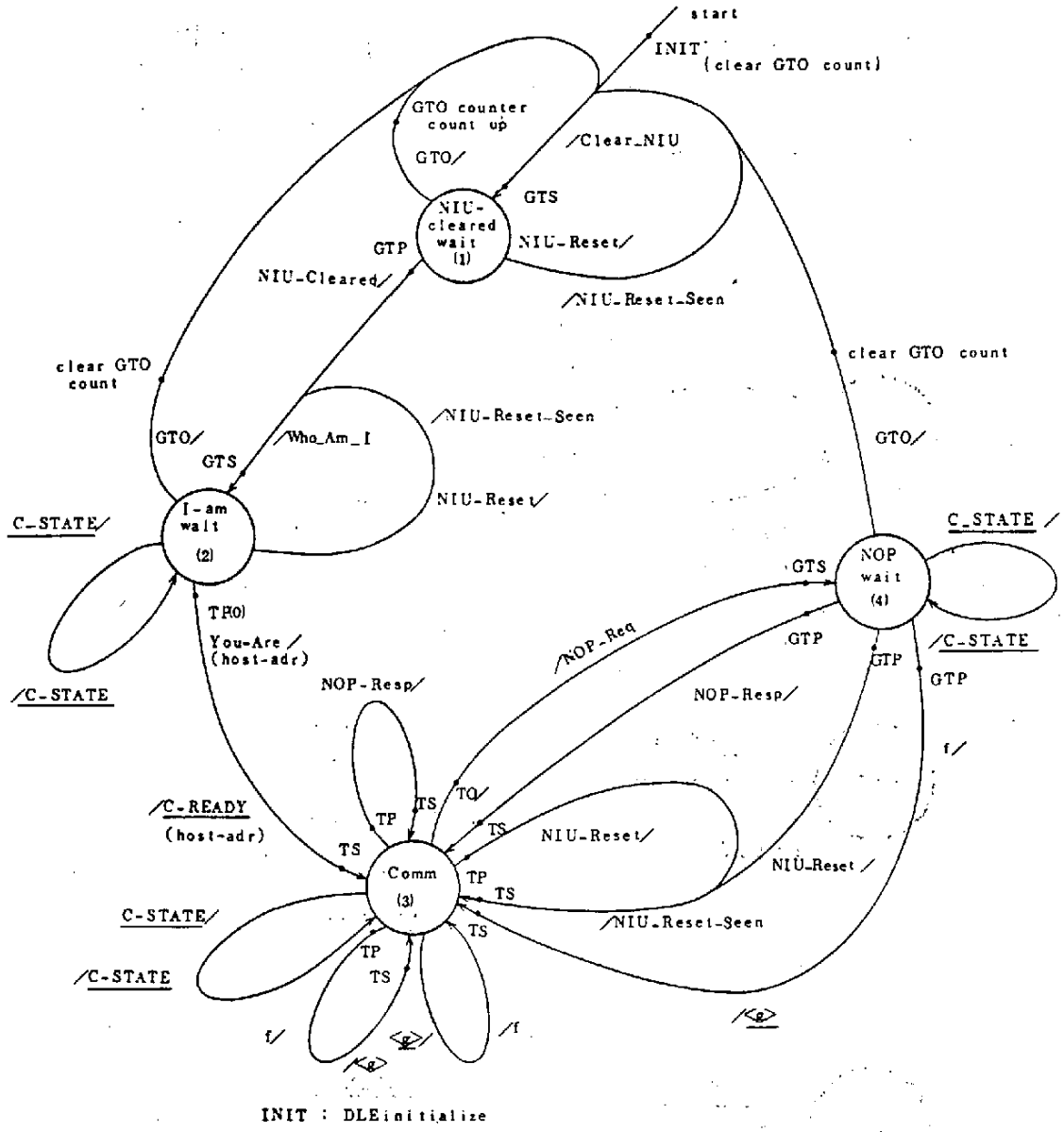
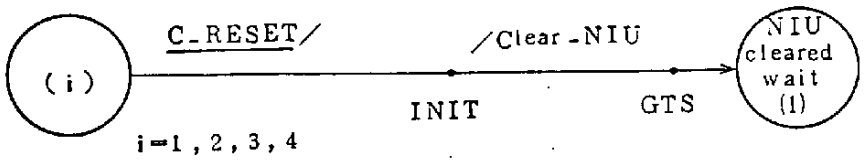
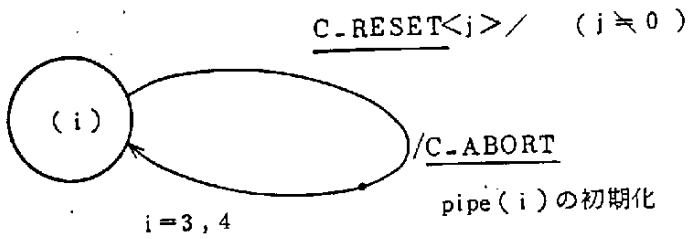
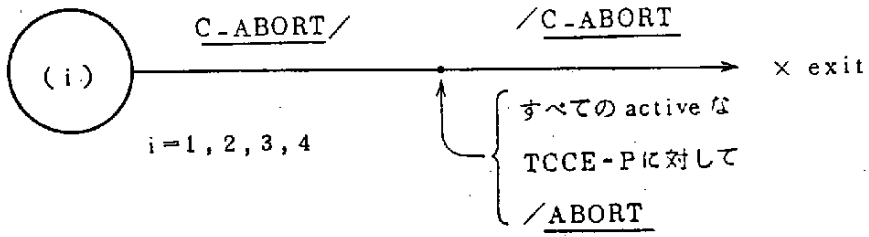


図3.2.1　ＤＬＥ状態遷移図(1／3)







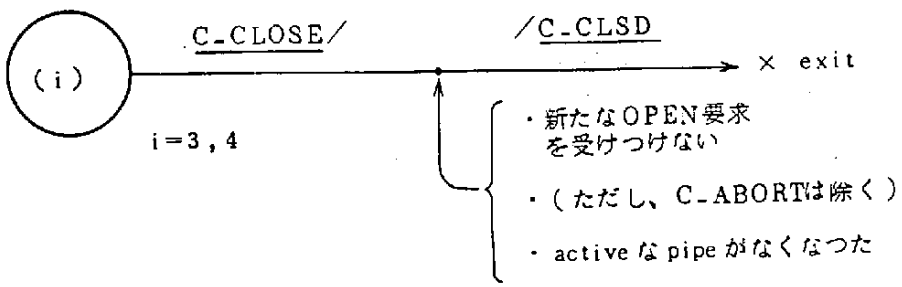
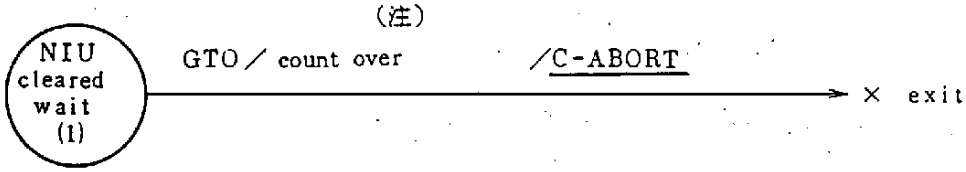


図3.2.2　DLE状態遷移図（2／3）



(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3.2.3　DLE状態遷移図（3／3）

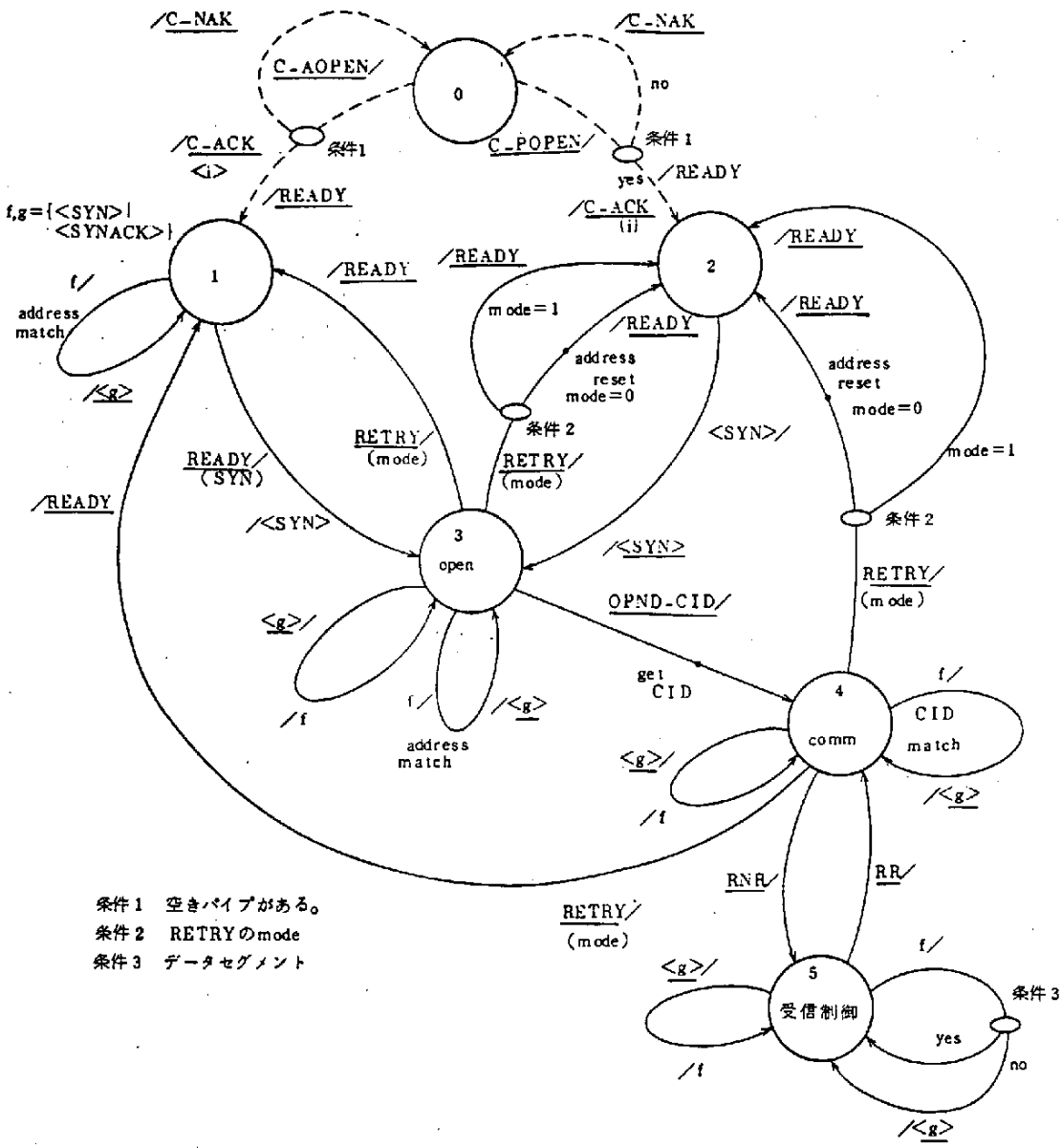


図 3.2 4 パス i の状態遷移図 (1／2)

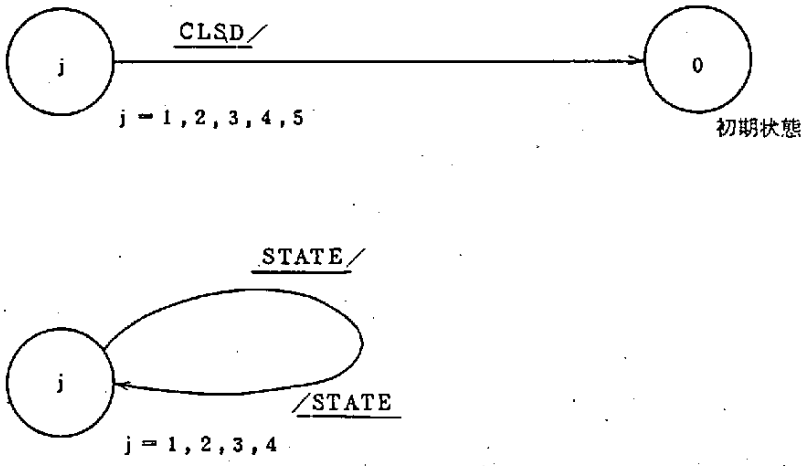
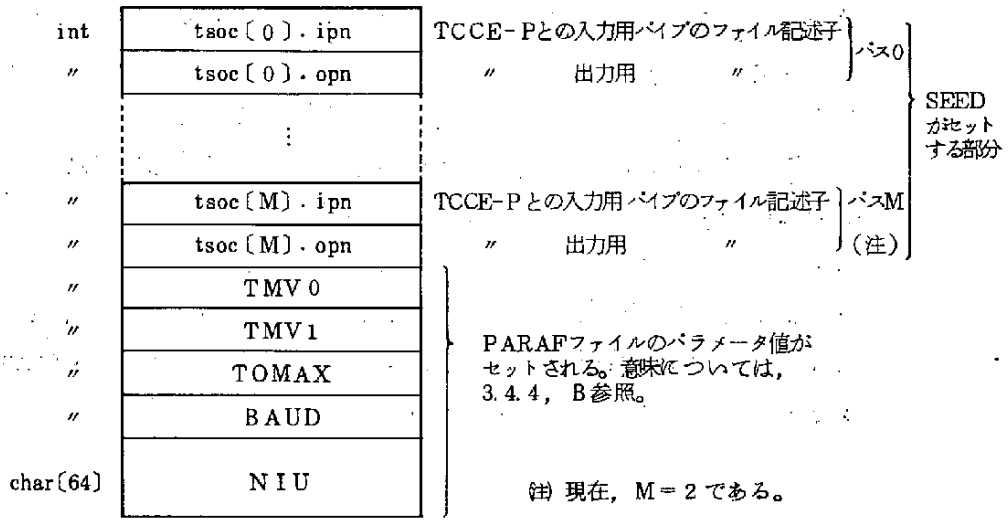


図3.2.5　パスiの状態遷移図（2／2）



int ：整数型（ SUN-2：4byte　U-1200：2byte ）

char〔n〕：n個の文字型の配列（ n byte ）

図3.2.6　DLIFインターフェースファイルフォーマット

D. 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。

① パスを管理し，群の開設要求に対しパスを配当する。

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。

①の処理は，ＴＣＣＥ－Ｓとの間でパス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，未使用パスを選定し，そのパス番号 i（$i=1$，2）をＴＣＣＥ－Ｓにフレーム C－ＡＣＫに格納して通知する。このパス 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，未使用パスがない場合には，フレームＣ－ＮＡＫにより開設要求を拒否する。ＤＬＥはパスの配当時，パス管理テーブル〔図3.27〕を作成する。このパス管理テーブルにより，群通信とパスの対応を図る。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全てのパケット（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。

int	sstate	パスiの状態遷移番号
char〔12〕	addr〔0〕	群を構成する実体のソケット番号のリスト
	⋮	
char〔12〕	addr〔MEN−1〕	
int	en	群を構成する実体数
〃	myno	群内での自実体の番号
〃	type	'A'：Active Open　'P'：Passive Open
〃	mode	'0'：無記名　'1'：記名
char〔14〕	CID	群識別子

int：整数型（SUN−2：4byte，U−1200：2byte）
char〔n〕：n個の文字型の配列（n byte）

図3.27　パス管理テーブル

E. 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。

① 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3.2〕。

② ＴＣＣＥ−ＰとＥＤＬＥ（ＮＩＵ）との間でセグメントの送受をを行うためデータ用のＥＤＬフレーム“Here Is Data”（以下ＥＤＬ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。

①の処理は，ＤＬＥプロセス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ＥＤＬフレームを用いて行われる。

(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－Response”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。

3.3.6 TCCE－S

A. 概　要

ＴＣＣＥ－Ｓの役割は，ＤＤＢＥ－Ｐから受けた群開設要求に基づき，ＴＣＣＥ－Ｐプロセスを起動させることである。このため，ＤＬＥに群開設要求C-AOPEN又はＣ－POPENのフレームを送り，許可応答Ｃ－ＡＣＫを待つ。Ｃ－ＡＣＫフレームのデータ部には，起動されるＴＣＣＥ－Ｐが用いるパス番号がセットされている。ＴＣＣＥ－Ｓはこのパス番号とＴＳＩＦインターフェースファイルの内容とからＴＰＩＦインターフェースファイルを作成し，ＴＣＣＥ－Ｐを生成，起動させる。また，ＤＤＢＥ－Ｐに対しＣ－ＡＣＫを送信し，次の群開設要求を待つ。

一方，群開設要求に対して，ＤＬＥが拒否応答Ｃ－ＮＡＣを送ってきた場合には，ＴＣＣＥ－ＳもＤＤＢＥ－Ｐに対して拒否応答を行う。

またＴＣＣＥ－Ｓは，ＤＤＢＥ－Ｓから送られてくる管理用フレームに対して必要な処理をＤＬＥと協同して行なう。

B　状態遷移

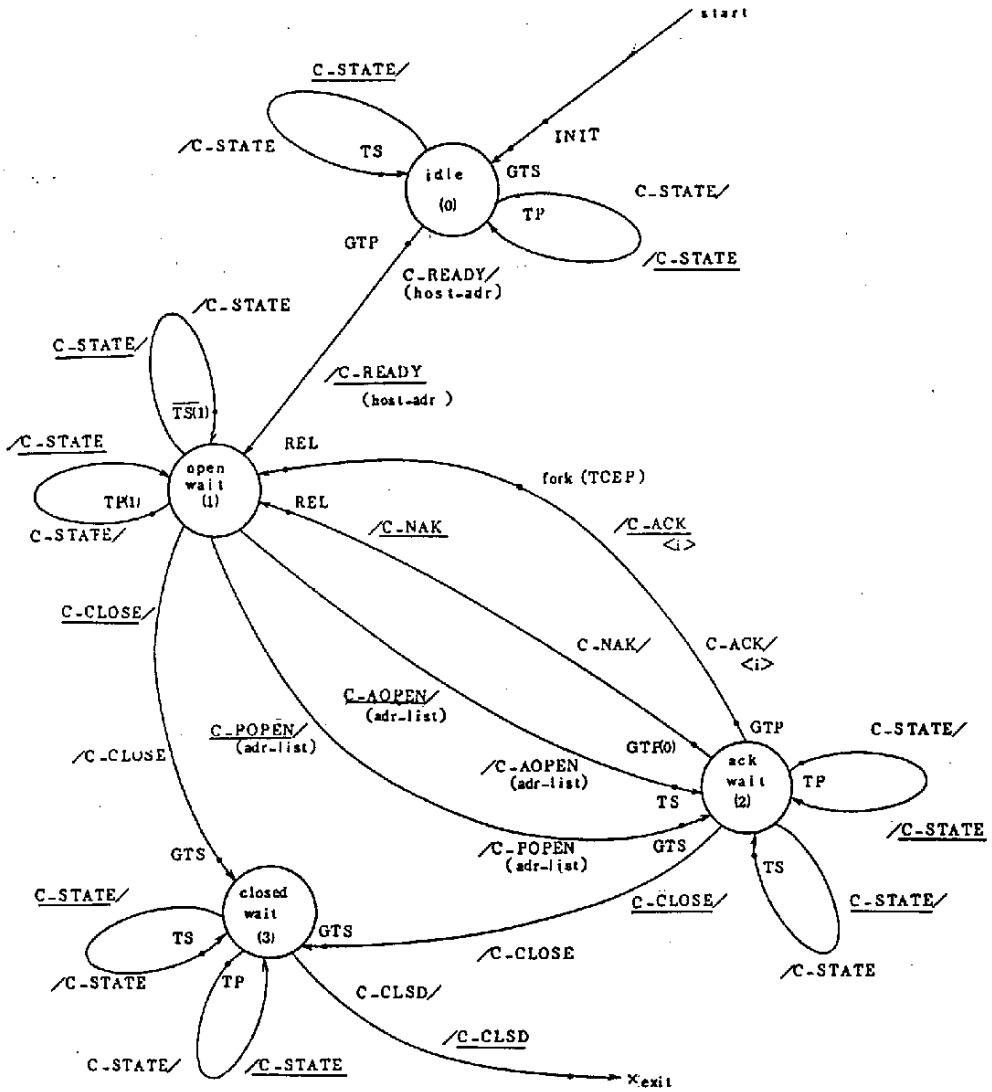
ＴＣＣＥ－Ｓの状態遷移図を図3.2８と図3.2９に示す。

C．インターフェースファイル

ＴＣＣＥ－Ｓは，ＳＥＥＤプロセスからＴＳＩＦインターフェースファイルを渡され，ＴＣＣＥ－ＰにＴＰＩＦインターフェースファイルを渡す。

ＴＳＩＦインターフェースファイルのフォーマットを図3.３０に示す。

ＴＰＩＦのフォーマットはＴＣＣＥ－Ｐの節を参照の事〔図3.４２〕。



INIT : TCES initialize

図 3.2 8　ＴＣＣＥ－Ｓ状態遷移図 (1)

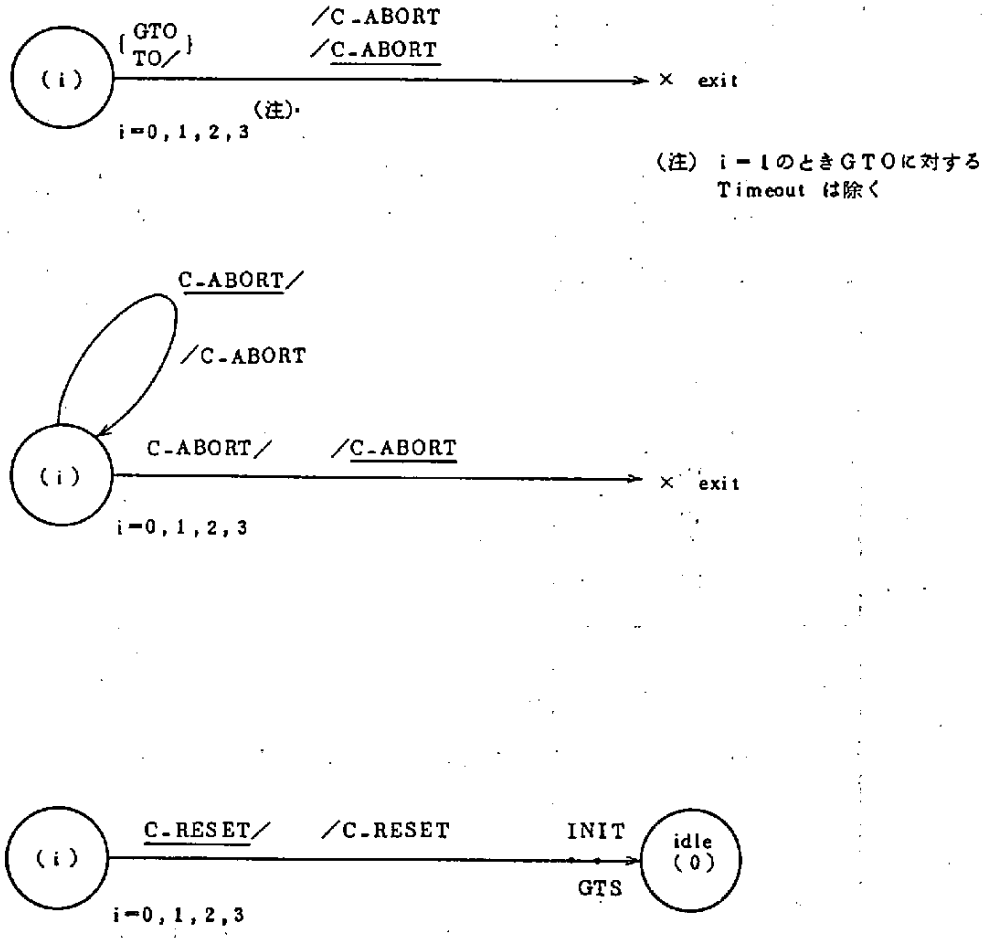
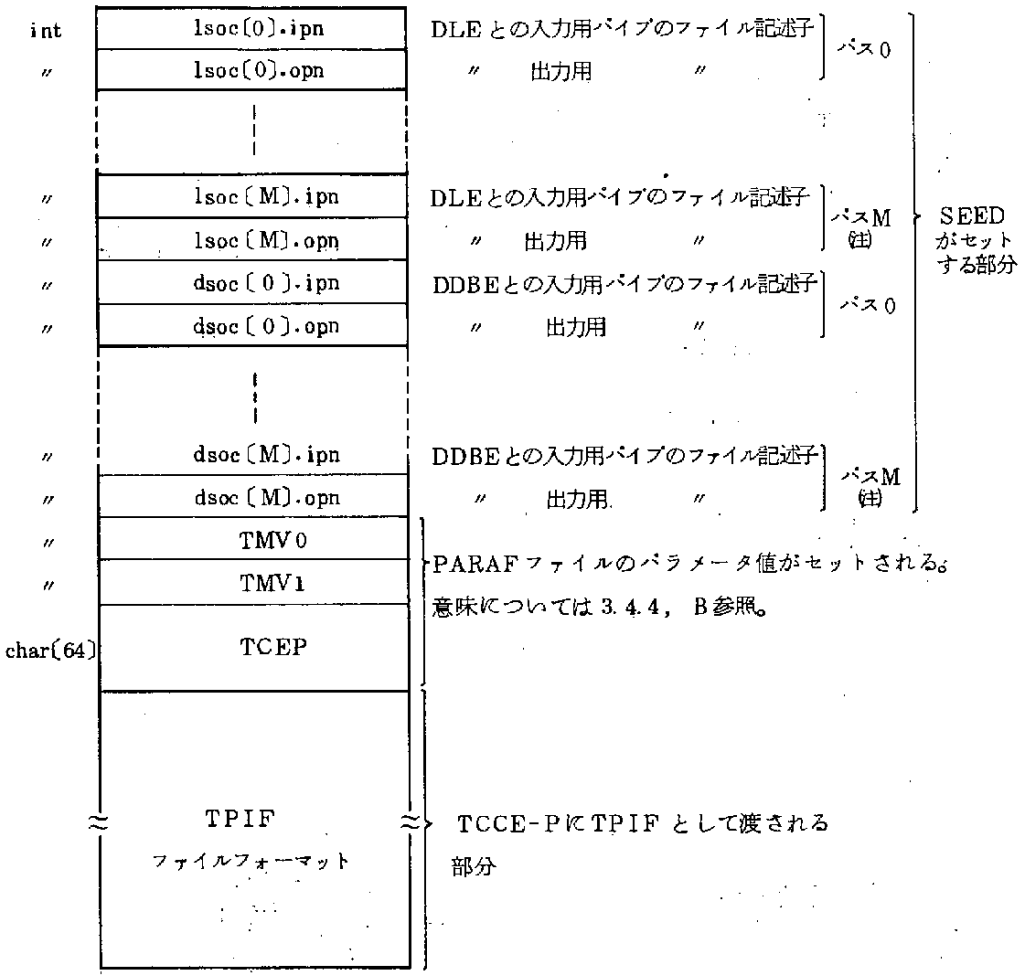


図 3.2.9　TCCE－S状態遷移図(2)



(注) 現在 M = 2

int : 整数型(SUN-2:4byte, U-1200:2byte)

char[n] : n個の文字列の配列(n byte)

図3.30 TSIFインターフェースファイルフォーマット

3.3.7 ＴＣＣＥ－Ｐ

A. 概　　要

ＴＣＣＥ－Ｐはトランスポート群制御プロトコル（ＴＣＣＰ）の機能のうち，

- 信頼放送性
- 順序性
- フロー制御

を実現し，また群の操作として，

- 群の開設
- 群の解除
- データ転送

をサポートする。

図3.3.1にＴＣＣＥ－Ｐの処理手順を，図3.3.2にモジュール関連図を示す。

ＴＣＣＥ－Ｐ内では，群開設処理，データ転送処理，群解除処理，ＲＥＪ（reject）処理，フロー制御処理，ＲＳＴ（reset）処理，およびABORT処理がプロトコル実現の為の基本処理単位となる。また，基本処理単位を円滑に処理する為に，ＡＣＫテーブル操作，キュー操作およびタイマー処理が存在する。

B. 状態遷移

ＴＣＣＥ－Ｐの状態遷移図を図3.3.3から図3.4.1に示す。記法については図3.2.0を参照されたい。

C. インターフェースファイル

ＴＣＣＥ－ＰはＴＣＣＥ－ＳからＴＰＩＦインターフェースファイルを渡される。図3.4.2にフォーマットを示す。ＴＣＣＥ－ＰはＴＰＩＦをread後，TPIFを削除して，新たなＴＣＣＥ－ＰをＴＣＣＥ－Ｓが起動出来る様にする。

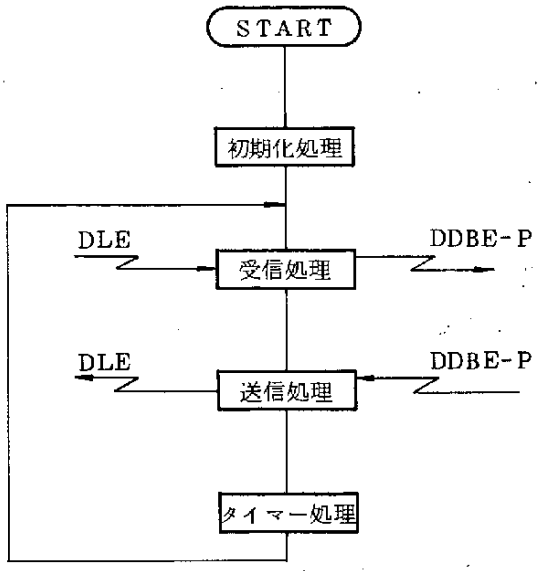


図3.3.1　TCCE－Pの処理手順

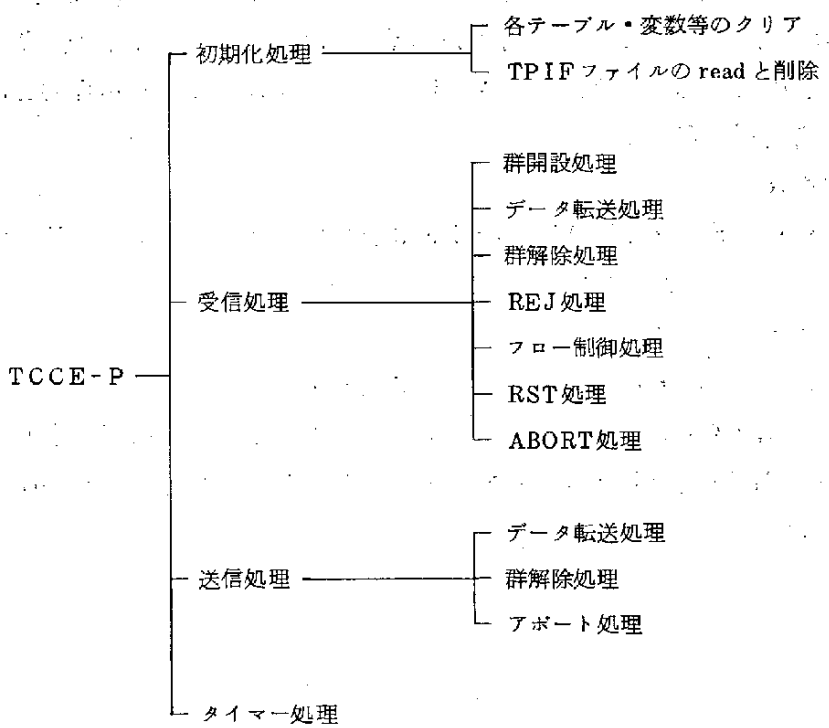


図3.3.2　TCCE－Pのモジュール関連図

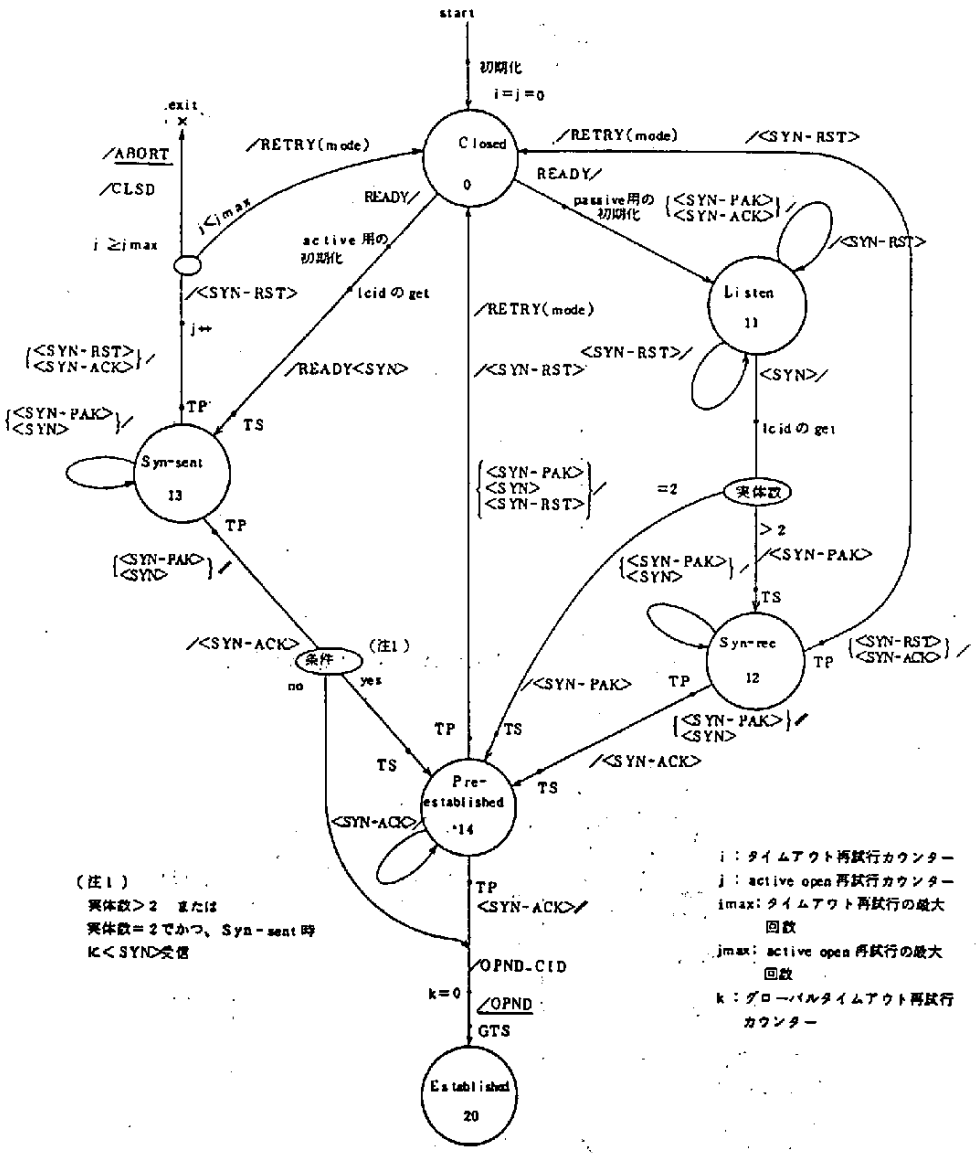


図 3.3.3：群開設処理状態遷移図(1)

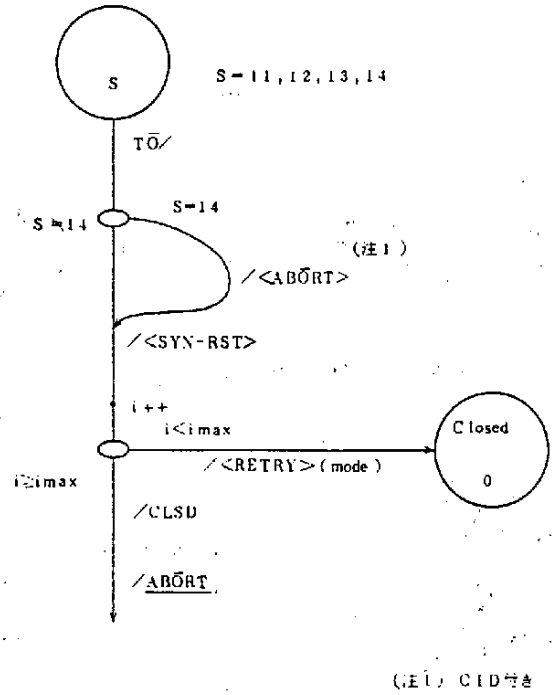


図 3.3 4　群開設処理状態遷移図(2)

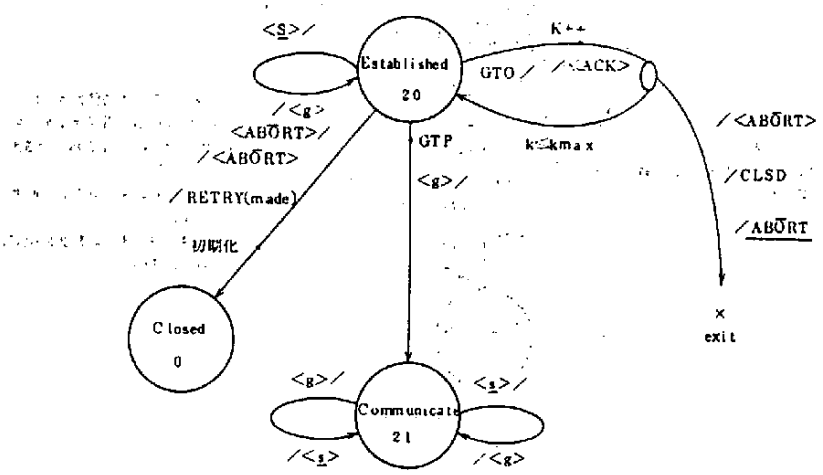


図 3.3 5　データ転送処理状態遷移図

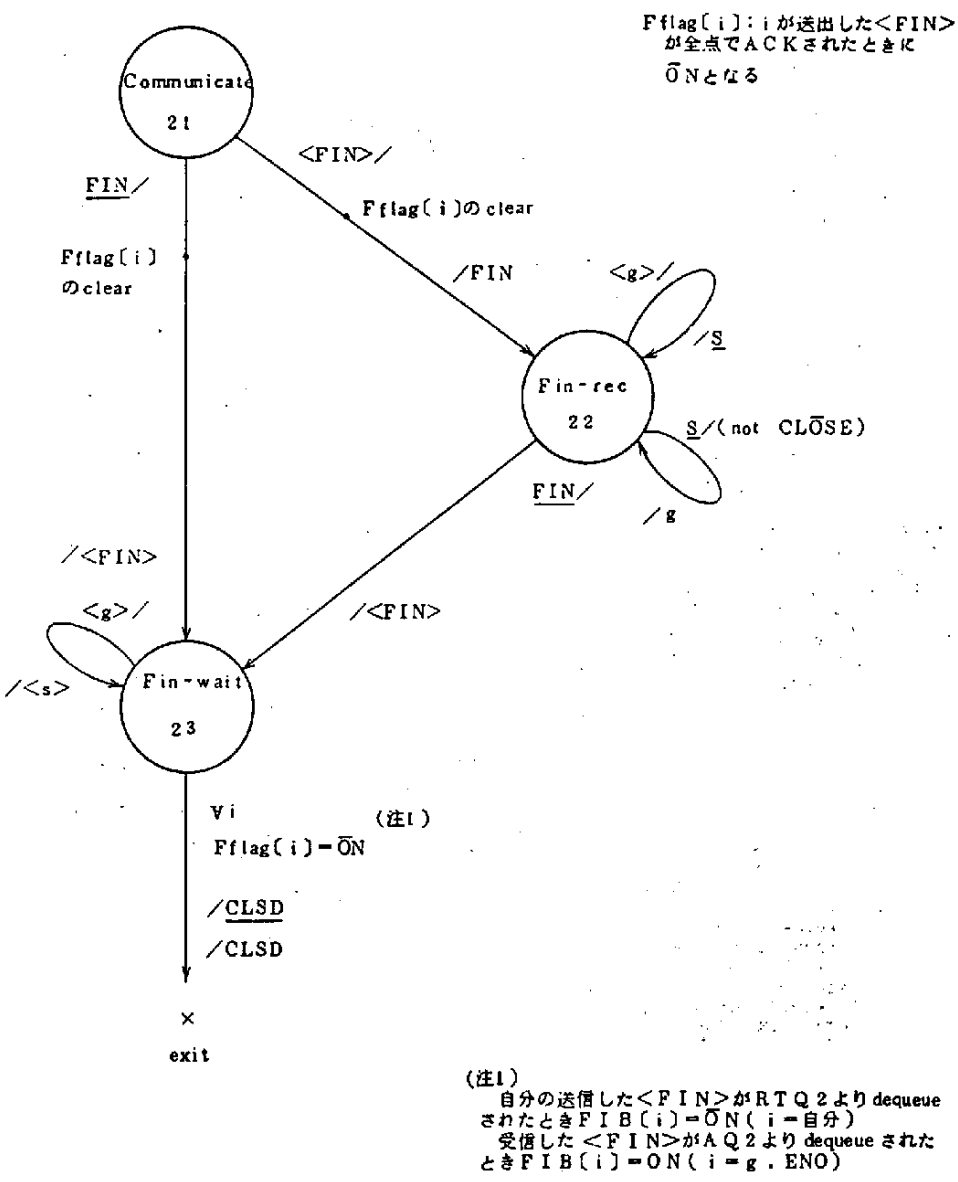
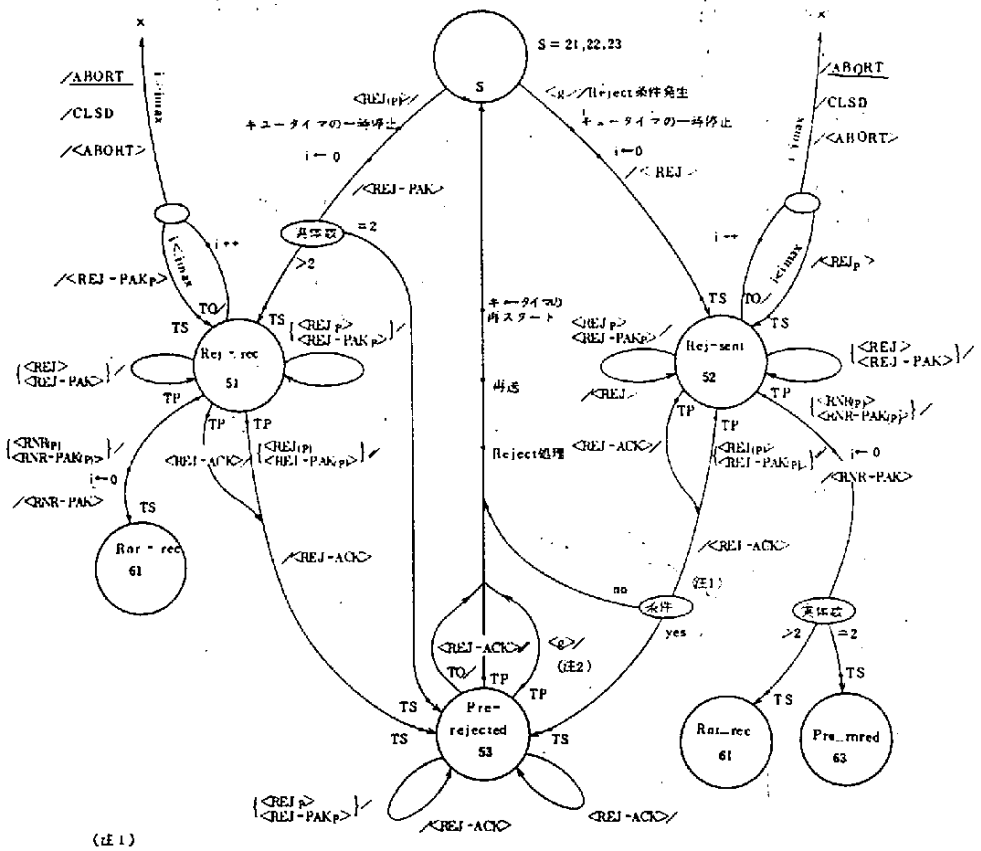


図 3.3 6　群解除処理状態遷移図



(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3.37　REJ処理状態遷移図

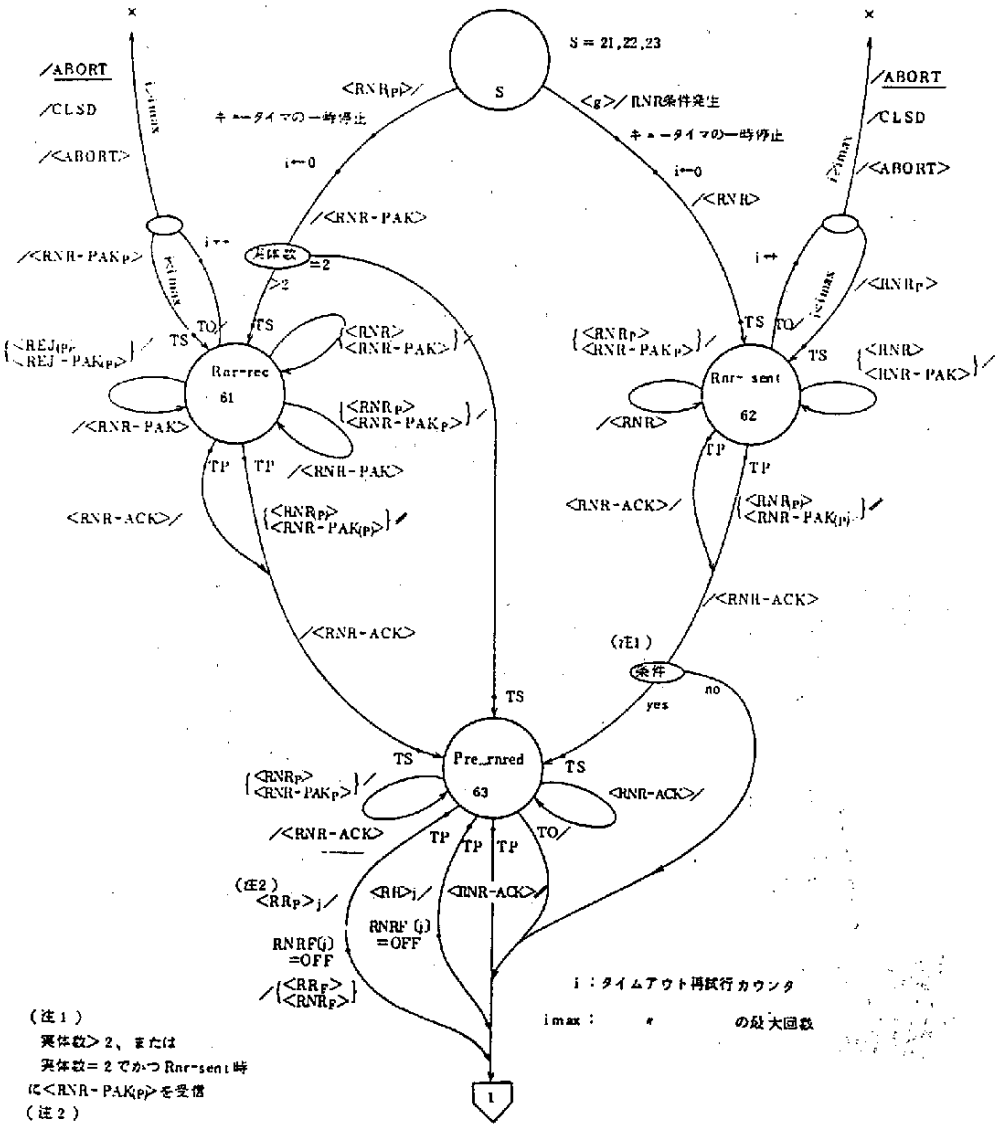


図3.3.8　フロー制御処理状態遷移図（1／2）

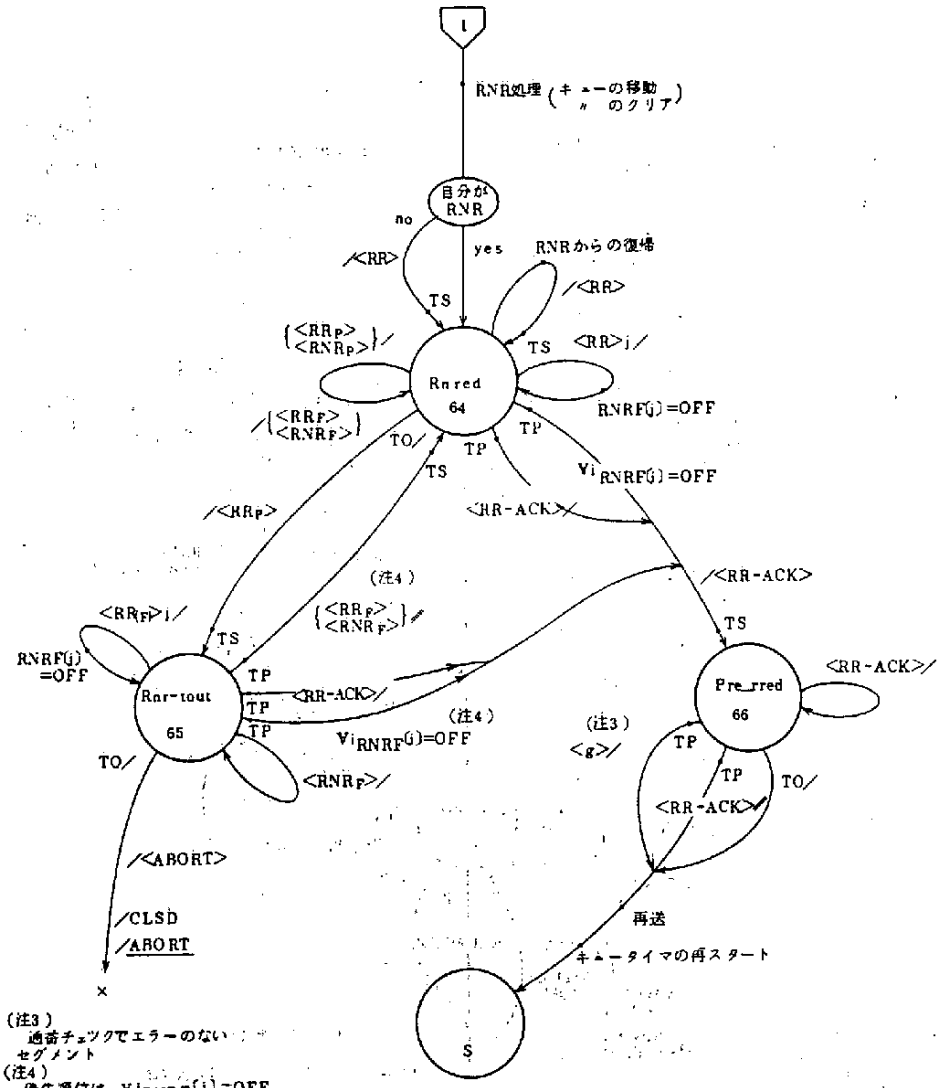


図 3.3 9　フロー制御処理状態遷移図（2／2）

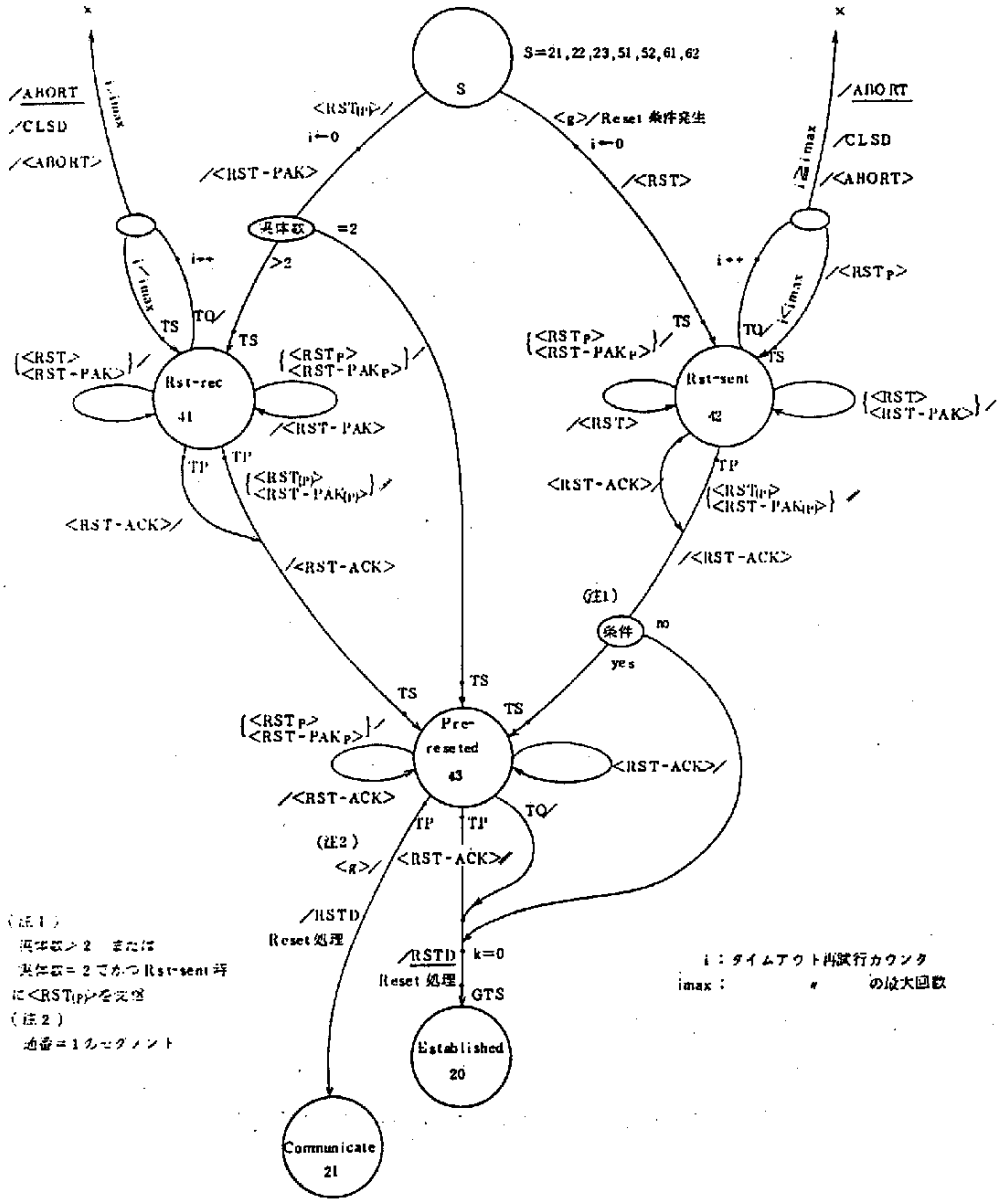


図3.40　ＲＳＴ処理状態遷移図

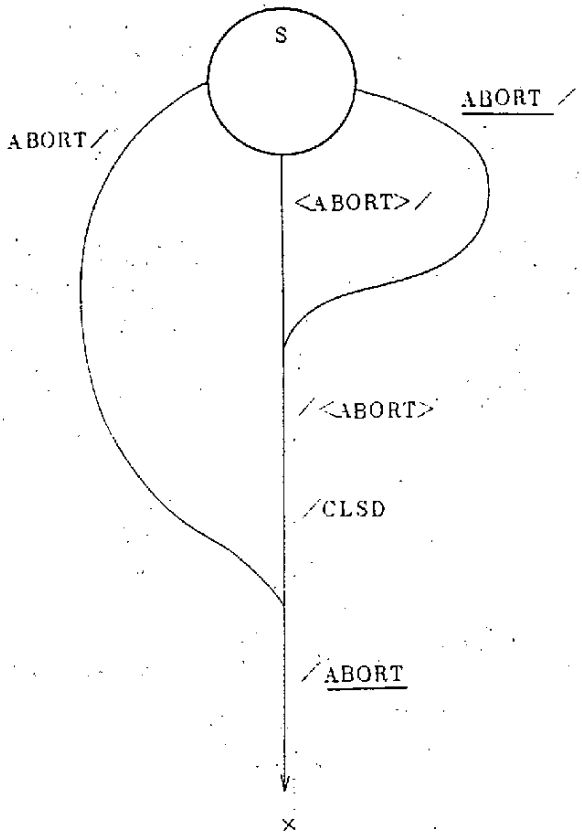
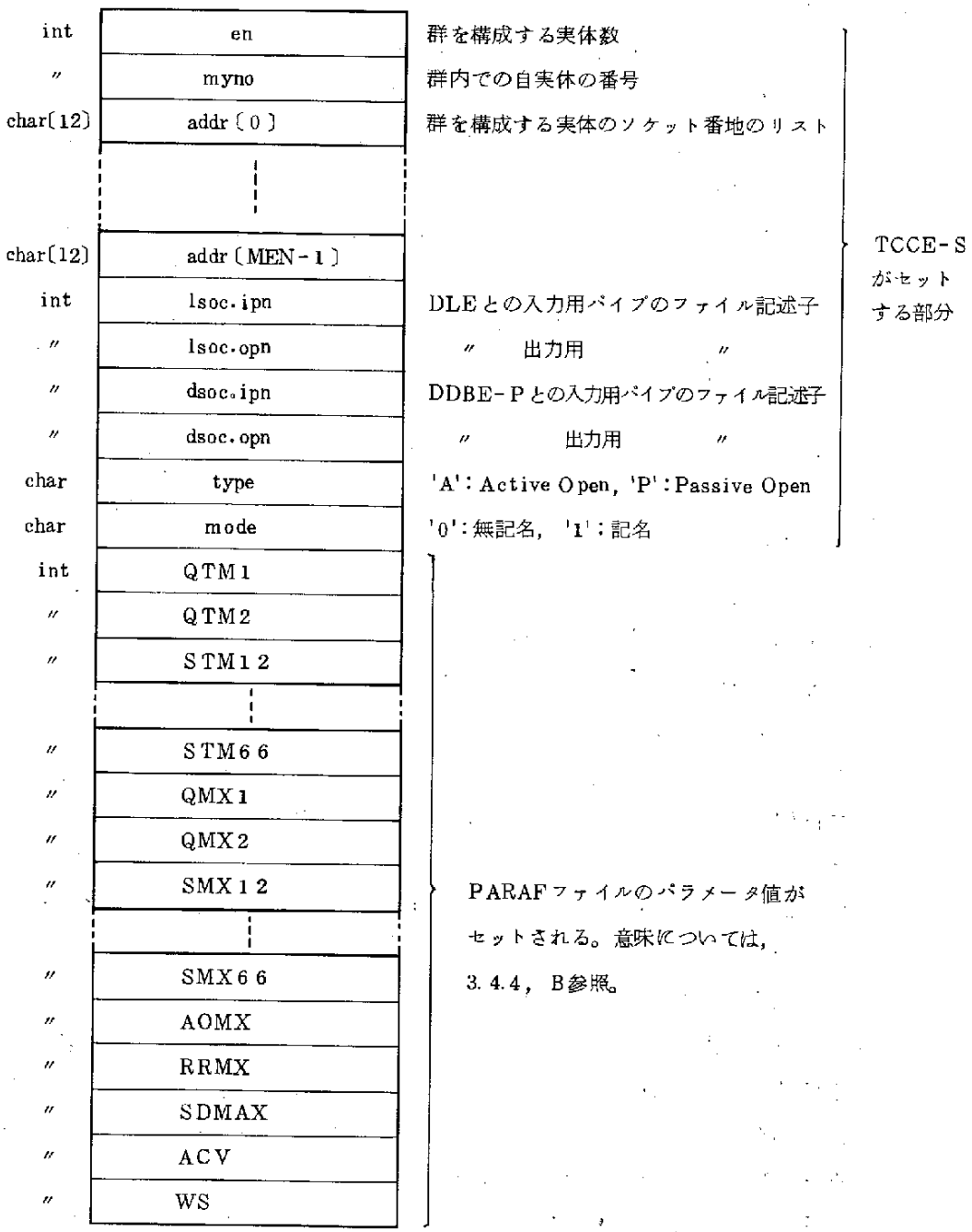


図 3.4 1　ABORT 処理状態遷移図



int　：整数型（SUN：4byte，U-1200：2byte）
char　：文字型（1 byte）
char〔n〕：n個の文字列

図3.42　TPIFインターフェースファイルのフォーマット

D. 群開設処理

(i) 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，Active Openで起動されたTCCE－P（activeな実体と呼ぶ）が送信する。
- ＳＹＮ－ＰＡＫは，Passive Openで起動された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。

(ii) 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。

(iii) 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。

(iv) 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（PARAFファイル内のAOMXパラメータ）が決めてあり，それ以上になるとTCCE－Pは終了（exit）する。

(v) 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，(iv)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。

E. 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－Ｐ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3.3１〕においてＤＤＢＥ－Ｐ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。

(iii) キュー操作については，Ｌ項を参照の事。

F. 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から捨てられる）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。

G. 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〔K. ＡＣＫテーブル操作参照〕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。

H. 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，フロー制御条件が発生した場合，又ＲＮＲを受信した場合に起る。

フロー制御条件とは，次の２つの場合をいう。

- セグメント受信時，ＡＱ１にキューイングする時にキューの為の領域が確保出来ない。
- セグメント送出時，ＲＴＱ１にキューイングする時にキューの為の領域が確保出来ない。

(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なるには，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解放される）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。

I. 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〔K.ＡＣＫテーブル操作参照〕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。

J. ABORT処理

(i) ABORT処理はABORTセグメントで行う。

(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〔D.群開設処理参照〕。

K. ＡＣＫテーブル操作

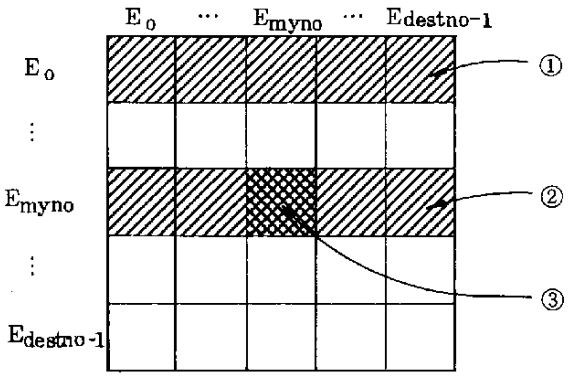
ＡＣＫテーブルとは，ＴＣＣＥ−Ｐ内でセグメントの通番を管理するためのテーブルである〔図3.4.3〕。

セグメントの通番とＡＣＫテーブルの操作について以下に述べる。

◦記　法

g.ack〔i〕	… セグメントの ack リスト。
ack-table	… ＡＣＫテーブル。
ack-copy	… ＲＥＪ，フロー制御処理に必要な，ＡＣＫテーブルの作業エリア。処理に入った時点で最初に ack-table よりコピーする。
ＥＮＯ	… セグメントｇを送出した実体の実体番号。
myno	… 自実体の実体番号。
ＲＥＪ	… ＲＥＪセグメント。
ＲＳＴ	… ＲＳＴ　〃
sendDLE(g)	… セグメントｇをＤＬＥに送出する関数。
＊	… 群内の全実体を意味する。
ＷＳ	… ウィンドウ幅。

ack-table



E_n ：n番目の実体
（ n = 0,…, destno-1）
E_{myno} ：自実体
myno ：自実体のEntity No
destno ：全実体数

ack-table (i,j) ：自実体E_{myno}が認識している，実体E_iが実体E_jから最後に受信した通番

ack-table $(0,j)$ ：自実体がE_0より受信したセグメントの最新のack（セグメントヘッダー内） ……①
$j=0,\cdots,$ destno-1

ack-table(myno,j) ：自実体が最後に送出したセグメントのackの内容 ……②
$j=0,\cdots,$destno-1

ack-table (myno, myno) ：自実体が最後に送出したセグメントの通番 ……③

ack-table (i,i) ：自実体がE_iより受信した最後のセグメントの通番
$i \neq$myno

図3.43　ACKテーブル

(i) セグメント＝<DATA>,<ACK>,<FIN>時

イ) セグメント受信時

① 受信したセグメントの通番が，正しい場合，受信したセグメントのackでACKテーブルを更新する。

```
if   g.ack〔ENO〕=ack-table〔ENO, ENO〕+1
     ack-table〔ENO, *〕=g.ack〔*〕;
```

② 再送のセグメントの場合，無視する。

```
if   g.ack〔ENO〕≦ack-table〔ENO, ENO〕
     ;
```

③ 受信したセグメントのあるack〔i〕が，ACKテーブルより大きい場合，差がウィンドウ幅内であれば，REJ処理を行う。

```
if   ack-table〔i,i〕+WS ≧ g.ack〔i〕 > ack-table〔i,i〕
     REJ.ack〔*〕= ack-table〔*,*〕;
     sendDLE(REJ);
     REJ process;
```

④ 受信したセグメントのあるack〔i〕が，ACKテーブルより大きい場合，差がウィンドウ幅内であれば，RST処理を行う。

```
if   g.ack〔i〕> ack-table〔i,i〕+WS
     RST.ack〔*〕=0;
     sendDLE(RST);
     RST process;
```

㊟ ③，④の条件時には，①，②が成立しないものとする。

ロ) セグメント送信時

① ack-tableの更新。

```
ack-table〔myno, *〕= ack-table〔*, *〕;
```

② セグメントの送出。

```
g.ack〔*〕=ack-table〔myno, *〕;
sendDLE(g);
```

(ii) セグメント＝＜REJ＞，＜RNR＞時

イ) セグメント送信時

① 自分が認識している各実体の通番をセットして送出。

```
g.ack〔*〕=ack-table〔myno, *〕;
sendDLE(g);
```

(iii) セグメント＝＜REJ−PAK＞，＜RNR−PAK＞時

イ) セグメント送信時

① ACKテーブルのコピーの更新，通番の小さい方を取る。

```
if   ack-copy〔i〕> g.ack〔i〕
     ack-copy〔i〕= g.ack〔i〕;
```

㊟ ここでgは，直前に受信した，〈REJ〉，〈REJ-PAK〉，〈RNR〉，〈RNR-PAK〉

② セグメントの送出。

g.ack〔＊〕＝ack-copy〔＊〕；

sendDLE(g)；

(iv) セグメント＝<REJ−ACK>，<RNR−ACK>，<RR>，<RR−ACK>時

イ） セグメント送信時

全実体で共通に認識している通番をセットして送出。

g.ack〔＊〕=ack-copy〔＊〕；

sendDLE(g)；

(v) RST処理後

ACKテーブルの全クリアー。

(vi) REJ処理，フロー制御処理後

ack-tableの更新。

for （i＝0；i＝destno-1；i＋＋）

ack-table〔i, ＊〕=ack-copy〔＊〕；

L. キュー操作

TCCE−Pでは2相Acknowledgementを実現する為に，AQ1，AQ2，SQ，RTQ1，RTQ2の5つのキューコントロールブロック（QCB）を持っており，データ転送処理中のデータ及びACKセグメントはいずれかのQCBにつながれている。

各QCBの役割は次のとおりである。

◦RTQ1

第1相〔図3.3参照〕の送信用QCB。DATA，FIN，ACKセグメント送出時にキューイングする。

◦RTQ2

第2相〔図3.3参照〕の送信用QCB。

◦AQ1

第1相の受信用QCB。

◦AQ2

第2相の受信用QCB。

◦SQ

Ack〔図3.3参照〕されたセグメント用のQCB。

図3.44にQCBの構造を，図3.45にキューの構造を示す。

以下に，キュー操作について述べる。

(i) セグメント＝<DATA>，<ACK>，<FIN>時

イ) セグメント受信時

① 受信セグメントｇをＡＱ１へキューイングする。

② ｇにより，ＡＱ１内でPreackされたセグメント（ｅとする)があれば，ＡＱ１からＡＱ２へ。

③ ②でｅがあった場合に，ＲＴＱ２の先頭のセグメントのaptrがｅをポイントしていれば，ＲＴＱ２よりはずす。同様の処理を，ｅをポイントしているセグメントが無くなるまで続ける。

④ ｇによりＲＴＱ１内で，Preackされたセグメント（ｆとする)があれば，ＲＴＱ１からＲＴＱ２へ。ｆのaptrでｇをポイントする。

⑤ ④でｆがあった場合に，ＡＱ２の先頭のセグメントのrptrがｆをポイントしていれば，ＡＱ２よりＳＱへ移す。同様の処理をｆをポイントしているセグメントが無くなるまで続ける。

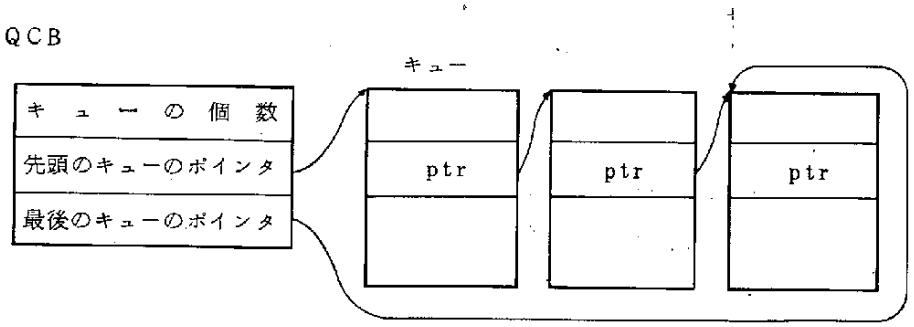
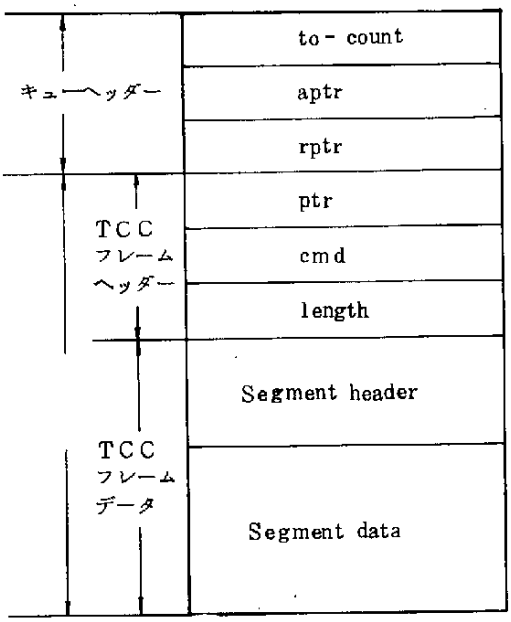


図 3.4 4　ＱＣＢの構造



∘ to-count

タイムアウト再試行用カウンター。このセグメントがタイムアウトになる度に増加させる。最大値になるとABORT処理を行う。

∘ aptr

RTQ1内のセグメントが，あるセグメントgの受信によりPreackされたときに，そのAQ1内のgのキューへのポインタである。

∘ rptr

セグメントgの送出時にAQ1内の受信セグメントのackをのせたときに，ackされた受信セグメントがRTQ1内のgのキューをポイントする為のポインタ。

図3.4.5　キューの構造

⑥　SQ内の先頭のセグメントについて，ストリームとして完成していれば（moreビットがONである。または，同じENOのセグメントでmoreビットONのものがある），DDBEに送信する。

ロ）セグメント送信時

①　送信するセグメントgをRTQ1へキューイングする。

②　gに，AQ1内の受信セグメントに対するackが乗っていれば，そのセグメントのrptrでgをポイントする。

(ii)　REJ，フロー制御処理後

①　AQ2よりAckされたセグメントをSQへ移す。

②　AQ1よりAckされたセグメントをSQへ移す。

③　RTQ2よりAckされたセグメントを捨てる。

④　RTQ1よりAckされたセグメントを捨てる。

⑤　RTQ1内に残ったセグメントがあれば，タイマーを再スタートして再送する。

(iii)　RST処理後

全ＱＣＢのクリアー。

M. タイマー処理

(i) タイマーの種類

タイマーは大きく次の２つに分れる。

・遷移の状態（ステート）にかかるタイマー

・セグメントにかかるタイマー

イ）遷移の状態にかかるタイマー

遷移の状態がCommunicate，Fin-rec，Fin-wait 以外のときにかかるタイマーである。タイマースタート，ストップのタイミングは3.4.7，B項を参照の事。

Established でかかるタイマーは，グローバルタイマーと呼ばれ，他のタイマー値よりも長い。

ロ）セグメントにかかるタイマー

キューイングされるセグメント，すなわち<DATA>,<ACK>,<FIN>については，セグメント自身にタイマーがかかる。

タイマーの種類は次の２通りである。

① ＲＴＱ１のセグメントにかかるタイマー

② ＲＴＱ２，ＡＱ２の　〃

ただし，①<②である。

(ii) タイムアウト操作

タイムアウト時の操作について述べる。

イ）遷移の状態にかかるタイマーのタイムアウト時

3.3.6，B項を参照の事。

ロ）セグメントにかかるタイマーのタイムアウト時

○ＲＴＱ１のセグメントのタイムアウト

セグメントの再送。

タイマーの再スタート。

○ＲＴＱ２のセグメントのタイムアウト

ＲＴＱ２よりはずす。

○ＡＱ２のセグメントのタイムアウト

ＡＱ２からＳＱへ移す。

3.3.8 ＤＤＢＥ－Ｓ

A. 概　　要

ＤＤＢＥ－Ｓは，利用者からの要求に基づいてＤＤＢＥ－Ｐプロセスを起動する事が主な目的である。ＤＤＢＥ－ＳはＳＥＥＤプロセスによって起動されると，ＤＳＩＦインタフェースファイルを read し，その後コンソールに対してＤＤＢＥ－Ｐプロセスのパス名を求めるプロンプトを表示する。

×××××　DDBEP？

利用者からの入力があると，ＤＰＩＦインターフェースファイルを作成し，その後ＤＤＢＥ－Ｐを起動する。

ＤＰＩＦインターフェースファイルは，ＤＤＢＥ－ＳからＤＤＢＥ－Ｐへパラメータを渡す目的の他に，両者の間でパス０の使用に関して排他的制御を行う目的で使われる。ＤＰＩＦインターフェースファイルが存在している間は，パス０の使用権はＤＤＢＥ－Ｐにある。よって，この間はＤＤＢＥ－Ｓはパス０への read／write を行なわないし，利用者に対して他のＤＤＢＥ－Ｐプロセスを求めるプロンプトの表示を控える。

B. 状態遷移

ＤＤＢＥ－Ｓの状態遷移図を図 3.4 6 に示す。

C. インターフェースファイル

ＤＤＢＥ－Ｓは，ＳＥＥＤプロセスからＤＳＩＦインターフェースファイルを渡され，ＤＤＢＥ－ＰにＤＰＩＦインターフェースファイルを渡す。両ファイルの内容は等しい。フォーマットを図 3.4 7 に示す。

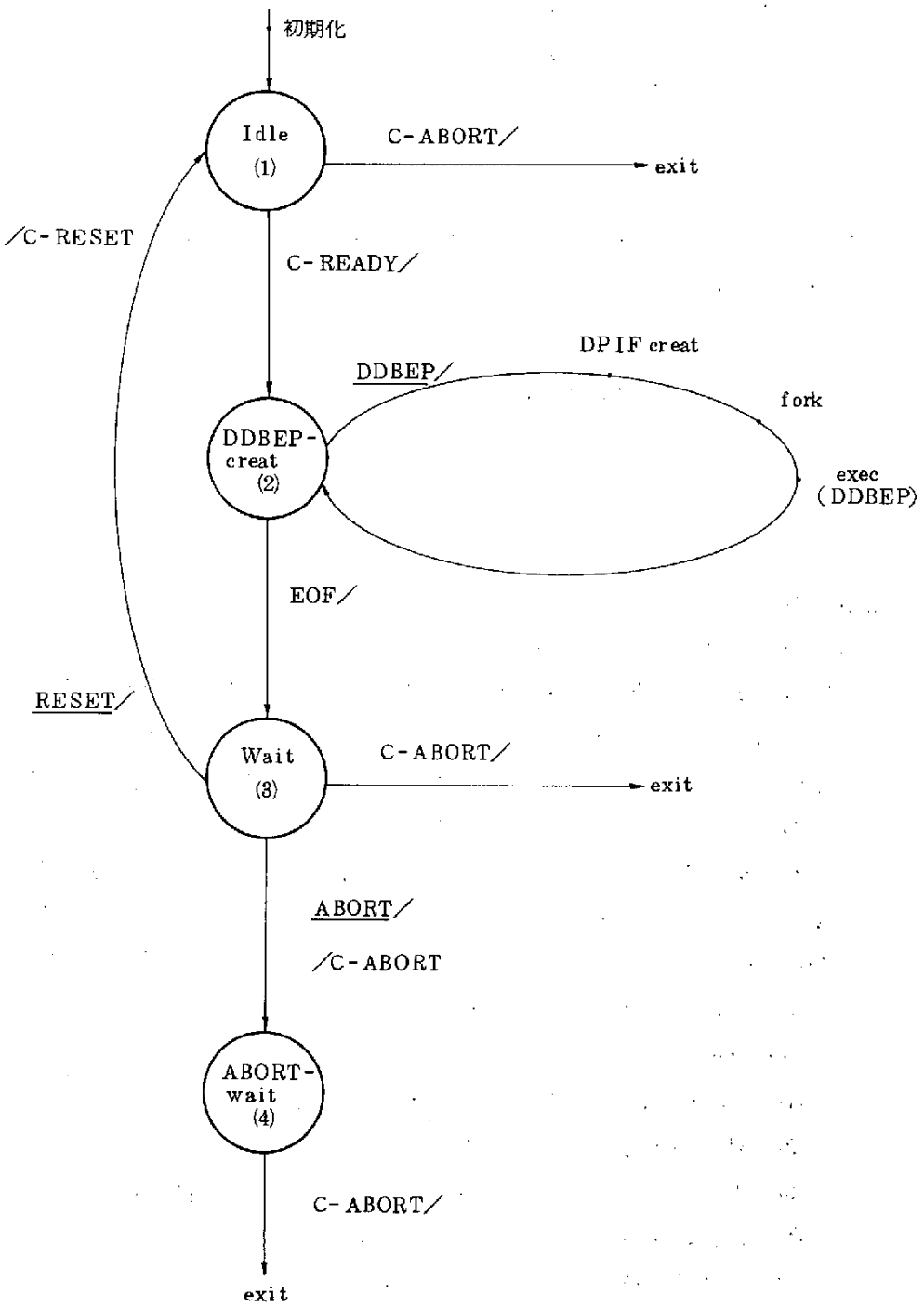
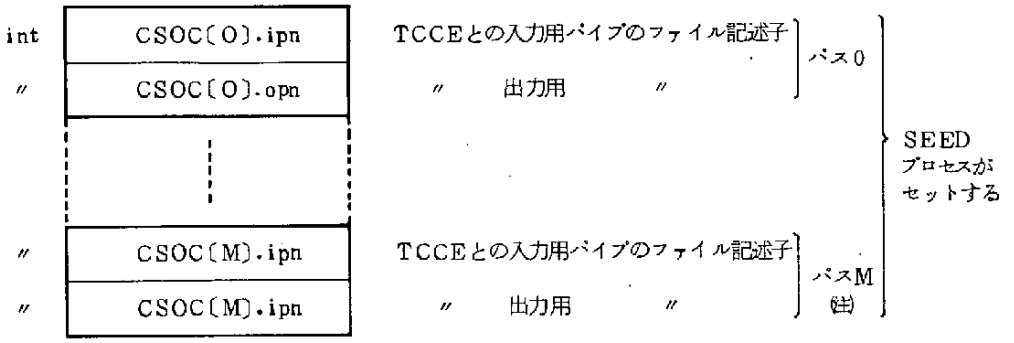


図 3.4 6　DDBE－S 状態遷移図



(注)現在M＝2

int：整数型（SUN-2：4byte，U-1200：2byte）

図3.47　DSIF，DPIFインターフェースファイルフォマット

3.3.9　ＤＤＢＥ－Ｐ

A. 概　　要

ＤＤＢ層の役割は，ＴＣＣ層が提供する群サービスを用いて上位層に対し分散型データベースシステム（ＤＤＢＳ）のサービスを行う事である。この項で述べるＤＤＢＥ－Ｐプロセスは，この役割を実現したＴＣＣ層の１つのアプリケーション例である。

DDBE－Pは機能により，ＵＤＢ（user ＤＢ）プロセスとＳＤＢ（server ＤＢ）プロセスとに分られる。ＵＤＢは群内にただ１つだけ存在するDDBE－Pプロセスであり，ＳＤＢは１つ又は複数存在するDDBE－Pプロセスである。図3.48にＵＤＢとＳＤＢの概念図を示す。

ＵＤＢの手順は，

①　Active Openを出し，ＳＤＢと群を開設する。

②　利用者から，トランザクションを入力する。

③　トランザクションを群内の全てのＳＤＢに送信する。

④　自分のトランザクションを実行する。

⑤　全サイト間でコミットメントを取る。

⑥　②からの繰り返し。

一方，ＳＤＢの手順は，

①　Passive Openを出す。

② ＵＤＢからのActive Openにより，群を開設する。

③ ＵＤＢからトランザクションを受信する。

④ 自分のトランザクションを実行する。

⑤ 全サイト間でコミットメントを取る。

⑥ ③からの繰り直し。

全サイト間のコミットメントには，２相コミットメント方式を用いる。

図3.49に３サイトによるDDBE−Pの手順を示す。

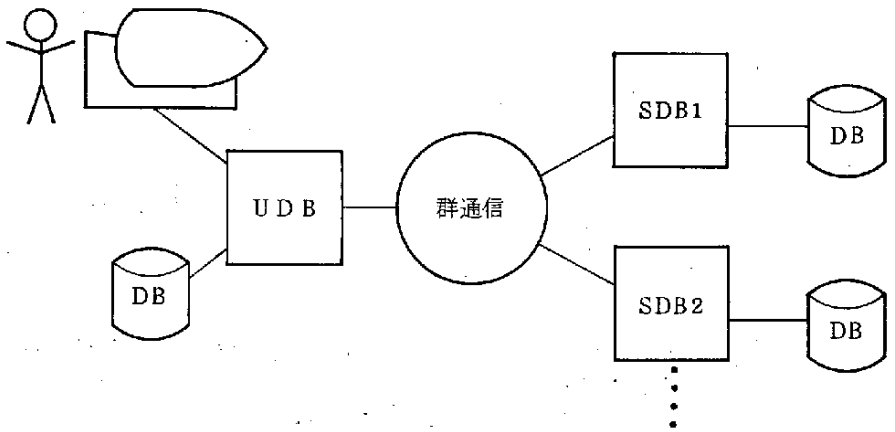


図3.48　ＵＤＢとＳＤＢ

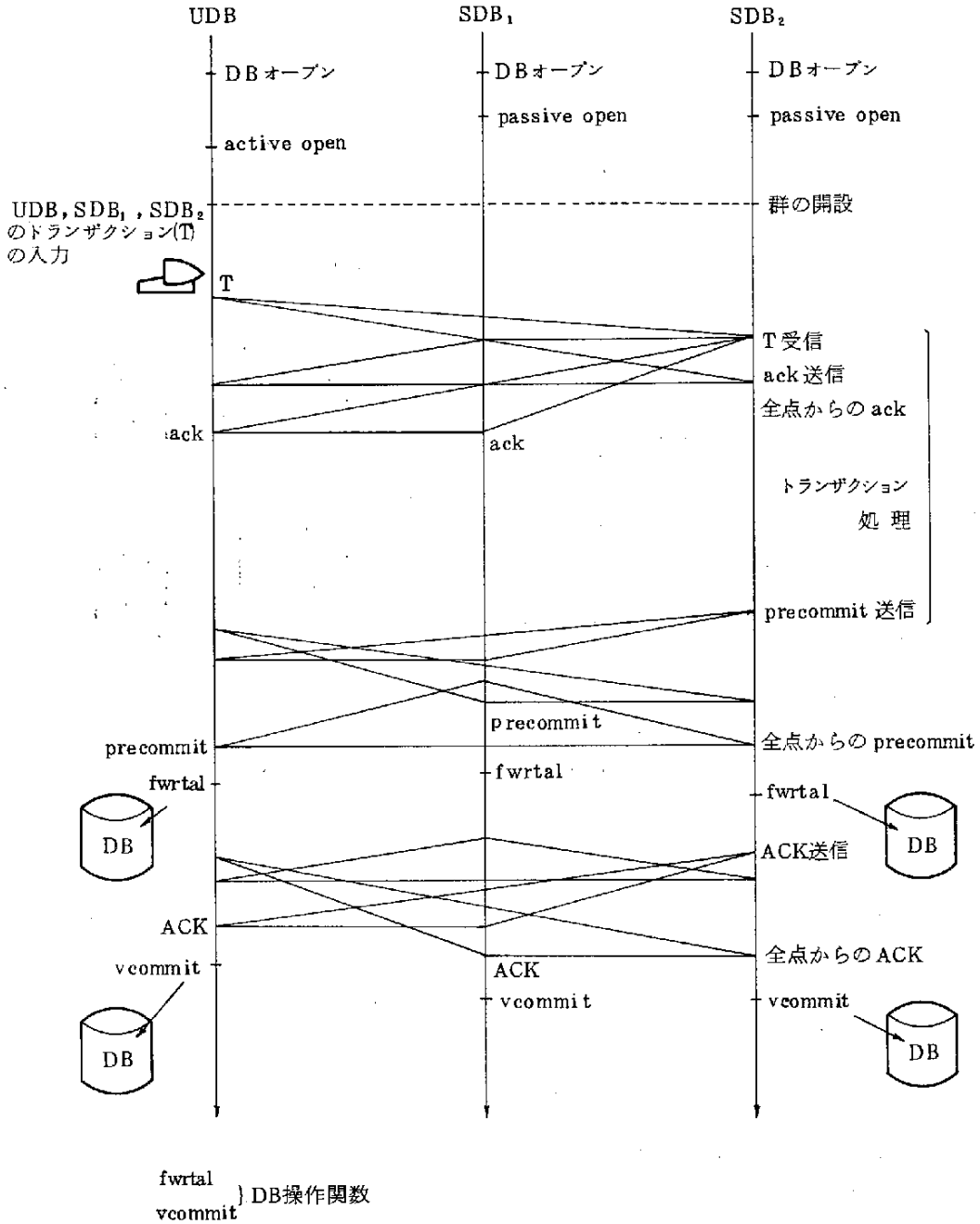


図3.49　DDBE−Pの手順

B. 状態遷移

UDBの状態遷移図を図3.50に，SDBの状態遷移図を図3.51に示す。

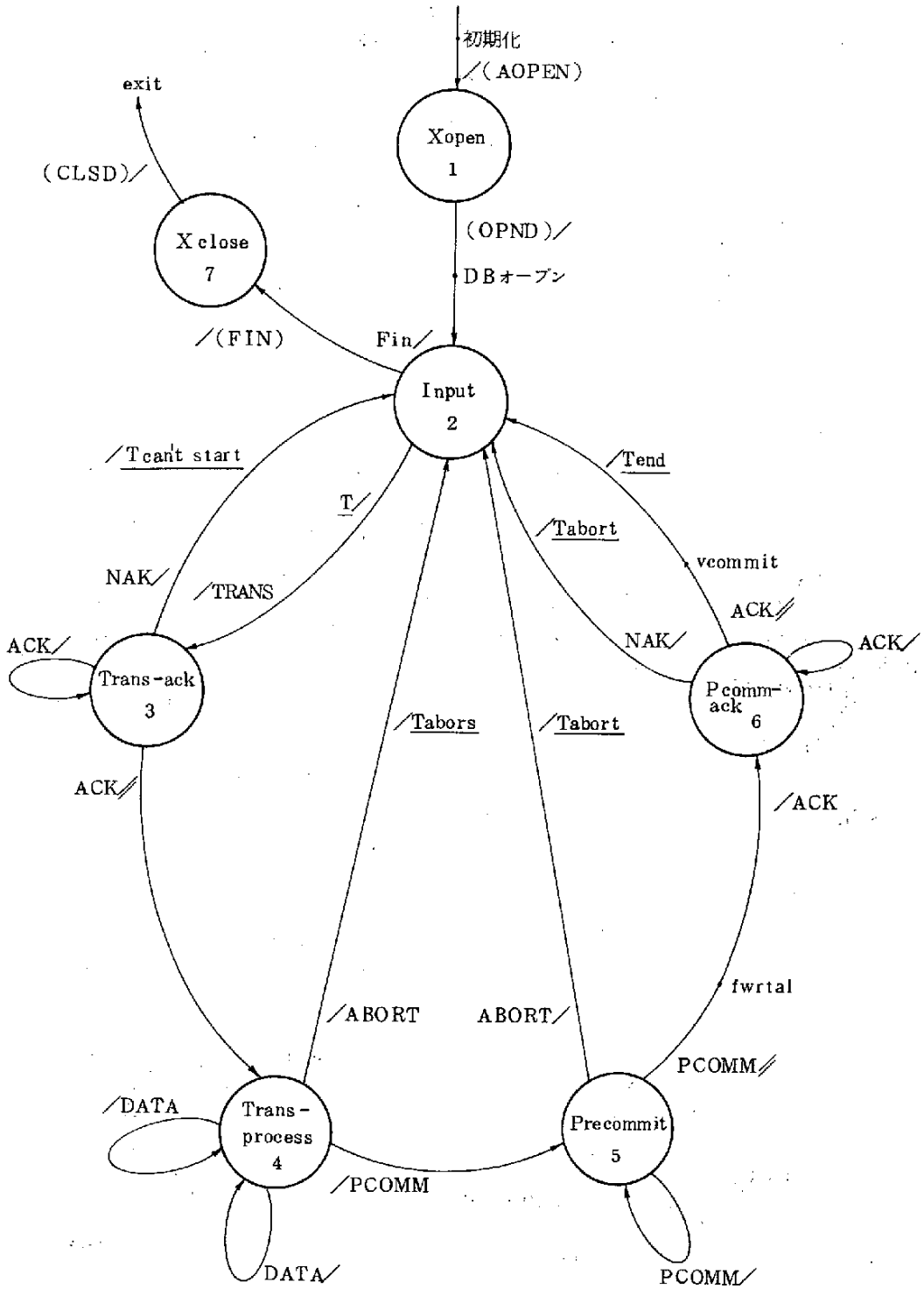


図3.50　ＵＤＢの状態遷移図

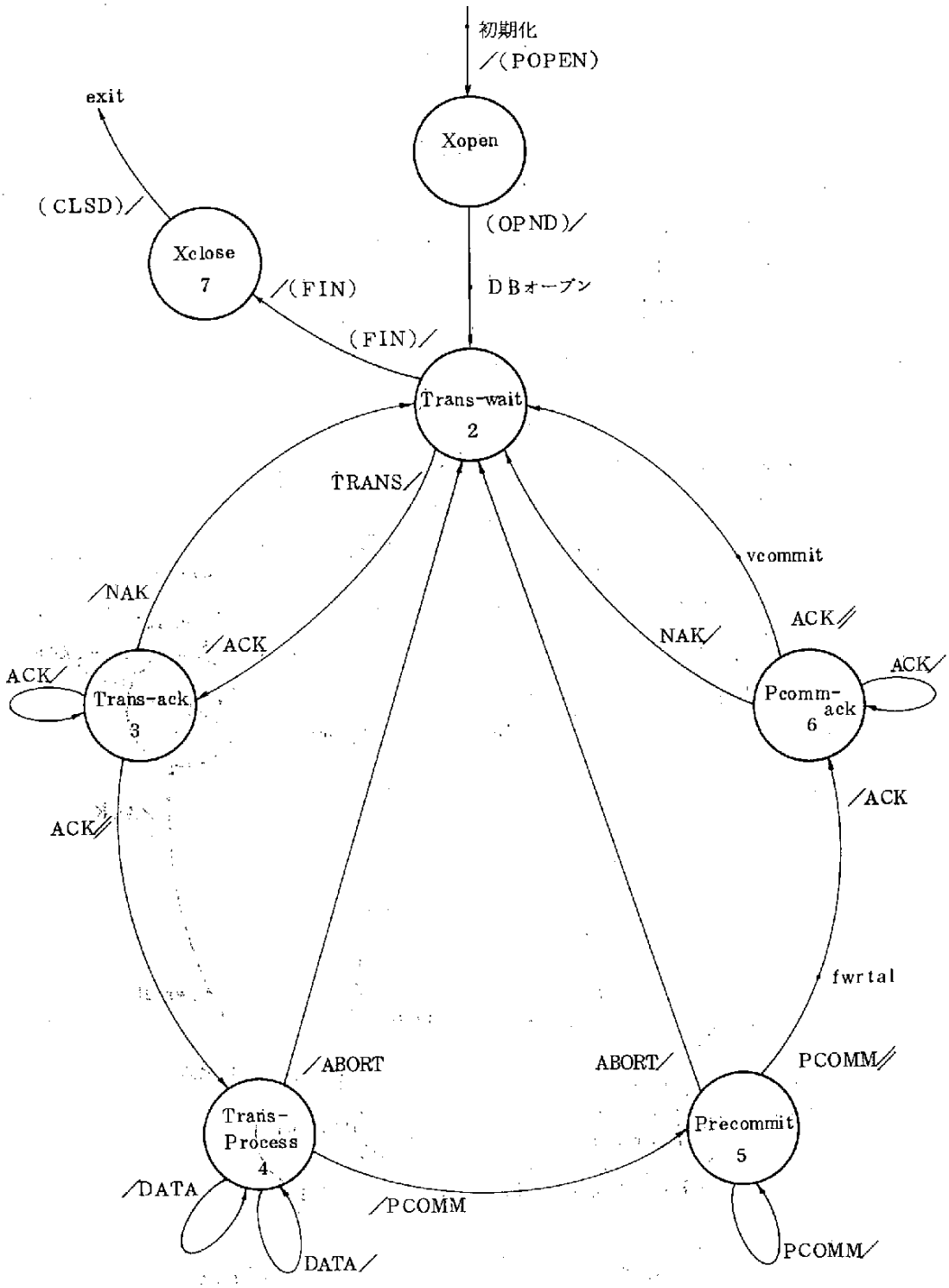


図3.5.1　SDBの状態遷移図

C. ストリームフォーマット

UDBとSDBで用いるストリームのフォーマットを図3.5 2に示す。各フィールドの意味は次のとおりである。

◦ストリーム長

このフィールドを含むストリームの長さ。

◦コマンド

ストリームの識別に用いる。コマンドには次のものがある。

TRANS … データがトランザクションである事を示す。

ACK … Acknowledge

NAK … Negative Acknowledge

PCOMM … Precommit

ABORT … アボート。

DATA … トランザクション以外のデータ。

◦Destination アドレス

送信先DDBE-Pのアドレス。（FFFF）$_{\times}$の場合は，ブロードキャストを意味する。

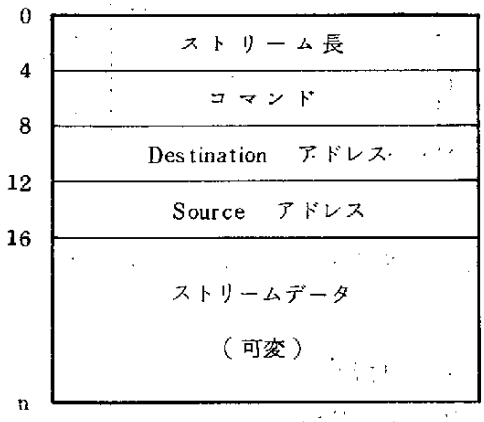


図3.5 2 ストリームフォーマット

∘ Source アドレス

送信先DDBE－Pのアドレス。

∘ ストリームデータ

コマンドがＴＲＡＮＳの場合にはトランスザクションデータが，コマンドがＤＡＴＡの場合には一般のデータが格納される。

D. トランザクションの構造

(i) ＵＤＢからＳＤＢに送信されるトランザクションのストリームフォーマットを図3.5 3に示す。

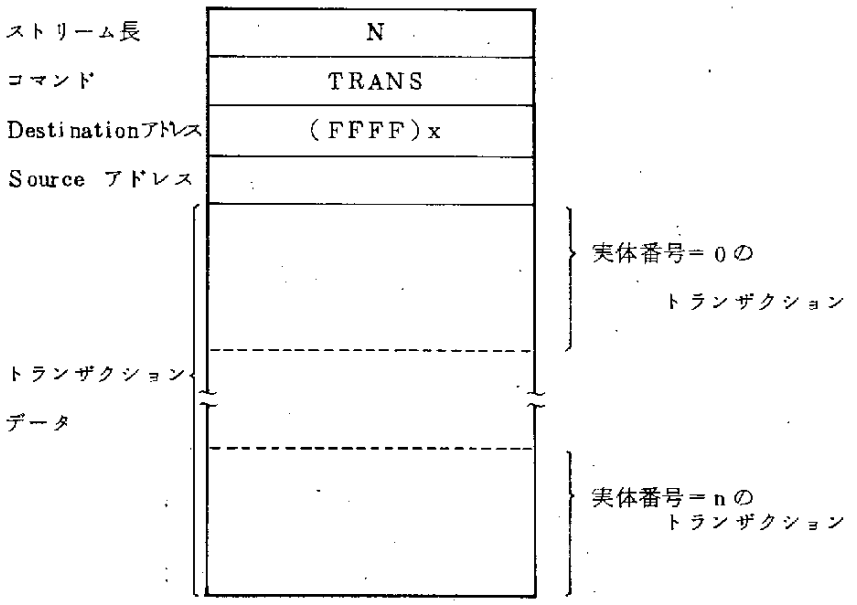
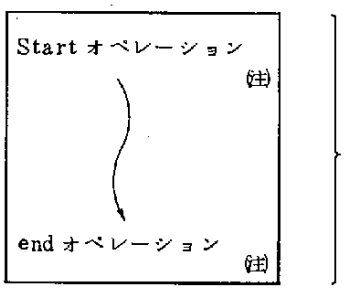


図3.5 3　トランザクション・ストリームのフォーマット

(ii) トランザクションの構成

1トランザクション＝Start オペレーションと end オペレーション
で囲まれたオペレーションの集合

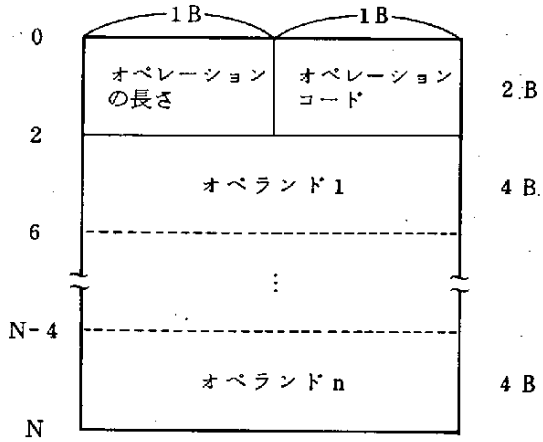


(注) ストリームに乗るのは，オペレーションの疑似コード

図3.5.4　トランザクションの単位

(iii) オペレーションのデータ構造

トランザクションを構成する各オペレーションは，次のデータ構造（疑似コード）で表わされる。

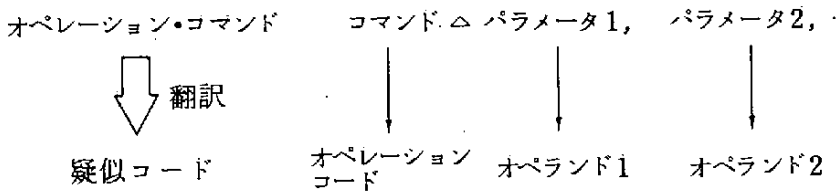


(注)

- オペレーションの長さは，各オペレーションで可変である。
  オペランドのタイプが文字型を含むオペレーションの時，
  オペレーションの長さ＝オペランドの数×4＋2
  (Byte)
  オペランドのタイプが文字型を含むオペレーションの時，
  文字列の大きさで可変

図3.5.5　オペレーションのデータ構造

(iv) オペレーション・コマンドとオペレーションのデータ構造（疑似コード）の関係



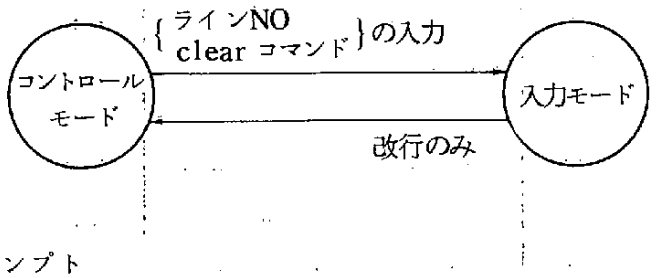
E. トランザクションの入力

(i) モード

画面からトランザクション・オペレーション（以下Tオペレーション）を入力する場合，UDBは2つのモードを持っている。

- コントロールモード…Tオペレーションのリスト，編集等のコマンドを受け付けるモード
- 入力モード ……Tオペレーションの入力を受け付けるモード

◦モードの遷移



◦プロンプト

“>△” …… コントロールモード時

“1.03△” …… 入力モード時

- 1.03 ― ラインNO＝3の入力を促している。
- 1 ― 実体番号が1のトランザクションである事を示す。

(ii) コントロール・コマンド

UDBがコントロール・モード時に入力可能なコマンド群。

① charge △ eno

カレント実体番号を eno に変更する。このときカレントラインNOは0にする。

② list △〔 n〔, m 〕〕

カレント実体番号のTオペレーションを表示する。

ex） list　　…全Tオペレーションの表示。 list ϕ, $ に等しい。

list △ 3　…ラインNO＝3の表示

list △ 5, 12 …ラインNO＝3～12の表示

list △ 5, $　…ラインNO＝5から最後までの表示

list △ , 20…最初（ラインNO＝0）から20までの表示

③ delete △ n〔, m 〕

カレント実体番号のTオペレーションのラインNO＝n～mを削除し，残りがある場合はつめる。

ex） Tオペレーション	delete△1, 3	delete△4	delete△2, $
1.00 A	1.00 A	1.00 A	1.00 A
1.01 B	1.01 E	1.01 B	1.01 B
1.02 C		1.02 C	
1.03 D		1.03 D	
1.04 E			

④ clear △ eno

実体番号がenoのTオペレーションを全てクリアし，入力モードに移る。

enoに"all"を指定したときは，全ての実体番号をTオペレーションをクリアし，カレント実体番号を0にして，入力モードに移る。

⑤ n

カレント実体番号のラインNO＝nで入力モードに移る。

⑥ exec

終了コマンド

⑦ dump △ file名

カレント実体番号のTオペレーションを全て，file名で指定されたファイルにダンプする。file名のファイルが存在しない場合には新規に作成する。このコマンドで作成したファイルは，loadコマンドの入力となる。

⑧ load △ file名

dumpコマンド，又はエディタで作成したTオペレーション列をカレント実体番号のTオペレーションとしてロードする。入力途中に不正なTオペレーションが存在した場合には，それまでの正しいものだけがロードされる。又，"#"で始まる行は，コメント行として無視される。コメント行は，fileのどこに存在しても良い。

⑨ sh

cshを呼び出す。

(iii) オペレーション・コマンド

○シンタックス

コマンド △ パラメータ1，パラメータ2,…

○パラメータのタイプ

変数　…最大3文字，最初の1文字はアルファベット

定数　…{ 整数 / 実数 / 文字："文字列"，スペース可能

ラインNO…0～n

イ) トランザクションの開始，終了

- start

  n：変数の個数

- end

ロ) メモリの確保

- tuple △ x，rid

  relation ID＝ridのtuple 1個分の領域を変数xに確保。

- var △ x，n

  nバイトの領域を変数xに確保し，領域を0クリアする。

ハ) 通信演算

- send △ x

  変数xのデータを送信する。

- recv △ x，eno

実体番号 eno からのストリームを変数 x に受け取る。

ニ) ＤＢのＩ／Ｏ演算

- fread △ x, rid, eof

  rid の先頭の tuple を compression しない形で変数 x にセットする。

- nread △ x, rid, eof

  rid の次の tuple を変数 x にセット。

- write △ x, rid

  変数 x のデータを rid に write する。

- del △ rid

  rid のカレントの tuple を削除する。

- delrel △ rid

  rid 内の全ての tuple の削除。

ホ) 定数のセット

- int △ x, n

  変数 x に整数値 n をセット。　　㊟ sizeof (n) = 4

- real △ x, m

  変数 x に実数値 m をセット。　　㊟ sizeof(m) = 4

- char △ x, " …… "

  変数 x に文字列 " … " をセット。　㊟ " は含まれない

ヘ) 分　岐

- goto △ ℓ

  行番号 ℓ に無条件分岐する。

- cst △ ℓ, n

  インジケータレジスタの値が n のとき，行番号 ℓ に分岐。

ト) 条　件　式

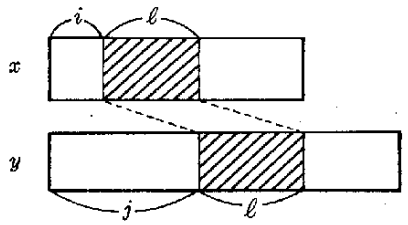
- if △ x, i, y, j, ℓ

  変数 x のオフセット i から長さ ℓ バイトのデータと，変数 y のオフセット j から長さ ℓ バイトのデータを比較して，結果をインジケータレジスタにセットする。

  $x > y$ のとき　インジケータスレジスタ = 1

  $x = y$ のとき　　〃　　= 0

$x < y$のとき　インジケータスレジスタ＝－1



(Ⅳ)　データの移動

・move △ x, i, y, j, ℓ

変数xのオフセットiから長さℓバイトのデータを，変数yのオフセットjから長さℓの領域にコピーする。

F．トランザクションの実行

ＵＤＢとＳＤＢの各プロセスは，自分用のトランザクション（オペレーションの疑似コード列）をインタープリターで実行する。

以下にインタープリターの仕様を示す。

(ｉ)　レジスタ

○ＢＲ（ベース・レジスタ）

トランザクション内の，自実体の start オペレーションをポイントする。

○ＰＣ（プログラム・カウンタ）

現在実行中の次のオペレーションの位置をポイントする。

ただし，分岐命令により値を変えられる可能性あり。

(注) ＢＲレジスタからの相対アドレス

○ＩＲ（インジケータ・レジスタ）

比較オペレーションにより，値を変えられる。又，ＣＳＴオペレーションは，ＩＲレジスタにより分岐先を決定する。

○ＳＲ（ステータス・レジスタ）

ＳＲ．ＣＣ……コンディション・コードを示す。

NORM	：	正常に実行中
END	：	正常終了（end オペレーションで終了）
ABEND	：	異常終了
ＩNTR	：	割込み中

ＳＲ．ＩＰ……SR．CC＝ＩＮＴＲ時にスタックされたＰＣの値

(ii) 実行手順

インタープリターにおける実行手順を図3.56に示す。

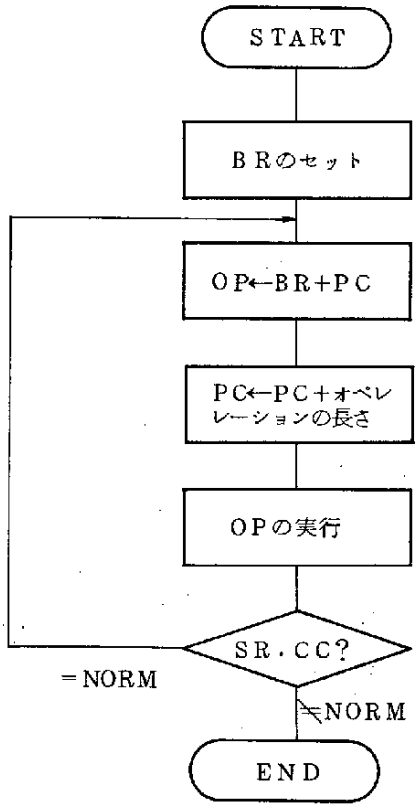


図3.56 実 行 手 順

(iii) 作業領域

トランザクション・オペレーション内の変数の領域は，実行時にダイナミックに確保される。領域はＬＷＡ（Local Working Area）と呼ばれ，LWACB（LWA Control Block）という配列で管理されている。LWACBの構造を図3.57に示す。

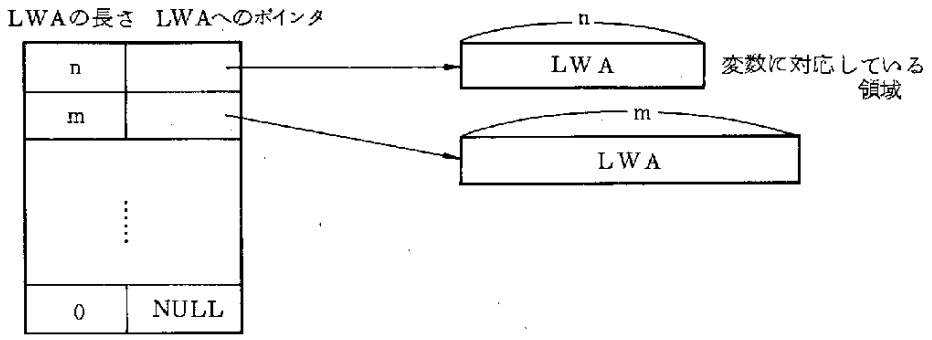


図3.57　LWACBの構造

G. 実　行　例

トランザクションの実行例を示す。ＵＤＢとＳＤＢの構成は図3.48のものとする。またＵＤＢの実体番号（ストリームヘッダー部のアドレスに相当する）を0，ＳＤＢ1の実体番号を1，ＳＤＢ2の実体番号を2とする。

図3.5.8にＵＤＢのＤＢのrelationIDが3の全tupleを，ＳＤＢ1のＤＢのrelationIDが1のrelationに，又ＳＤＢ2のＤＢのrelationIDが2のrelationにアペンドする側を示す。

```
                ≧ 0                                     入力モードへ切換え
                0.00    start
                0.01    tuple   X,3                     relation ID=3のtuple のエリアの確保
                0.02    char    EOF,"*eof"              終了時の同期用文字の定義
                0.03    fread   X,3,7                   先頭の tuple の読込み
UDBの           0.04    send    X                       送 信
トランザクション 0.05    nread   X,3,7                   次の tuple の読込み
                0.06    goto    4                       ループ
                0.07    send    EOF                     終了時
                0.08    end
                0.09                                    コマンドモードへ切換え
                ≧ change 1                              実体番号1のトランザクションの入力
                ≧ 0                                     入力モードへ切換え
                1.00    start
                1.01    tuple   Y,1                     relation ID=3のtuple のエリアの確保
                1.02    char    EOF,"*eof"
                1.03    recv    Y,0                     実体番号0からの読込み
SDB1の          1.04    if      Y,0,EOF,0,4             }終了条件の判定
トランザクション 1.05    cst     0,8                     }
                1.06    write   Y,1                     tuple のアペンド
                1.07    goto    3
                1.08    end
                1.09                                    コマンドモードへ切換え
                ≧ change 2                              実体番号2のトランザクションの入力
                ≧ 0                                     入力モードへ切換え
                2.00    start
                2.01    tuple   Z,2
                2.02    char    EOF,"*eof"
                2.03    recv    Z,0
SDB2の          2.04    if      Z,0,EOF,0,4
トランザクション 2.05    cst     0,8
                2.06    write   Z,2
                2.07    goto    3
                2.08    end
                2.09                                    コマンドモードへ切換え
                ≧ exec                                  実 行
```

下線部はＵＤＢからのプロンプト

図3.58 実 行 例

3.4 結果とまとめ

昨年度開発したトランスポート群制御プロトコル（ＴＣＣＰ）は，次の問題点をかかえていた。

- Net／OneのＮＩＵのメモリー不足（６４Ｋbyte）の為に，ＴＣＣＰをＮＩＵに実装出来ずホスト内に実現した為に，ＴＣＣ層だけでもＤＬＥ，ＴＣＣＥ－Ｓ，およびＴＣＣＥ－Ｐプロセスに分れ，ホストの処理効率が低下した。
- ホストとＮＩＵのインターフェースにＲＳ－２３２Ｃを使用した為，データ転送速度（9600bps）が遅く，Net／Oneの機能を十分に利用出来ない。
- ホスト側のRS-232Cのドライバーが端末用のドライバーである為に，大量のブロック転送に適さない（バッファサイズは256byte）。
- U－1200，SUN－2両ホスト間での移植性を考慮した為に，同じＵＮＩＸというＯＳでも機能的に劣るSystem Ⅲを搭載するＵ－１２００側に引きずられた。この結果，プロセス間通信としてパイプを，又非同期入出力機能としてもプロセスが常に状態センスをしなければならないという，非常に効率の悪いものを用いざるを得なかった。
- ＵＮＩＸ自体がＴＳＳを基本としている為に，リアルタイム処理に適していない。

本年度は，まず上記の問題点を解決すべく，

- Ｕ－１２００とＬＡＮとのインターフェースにＧＰ－ＩＢを用いて，データ転送の効率化と，端末用ドライバーのネックを解消する。
- ＬＡＮをNet／Oneから，メモリー容量も多くかつTCP／ＩＰプロトコルもサポートしているソリトンシステムズ社のものに変える。これにより，TCCPをＬＡＮ上に乗せられる。又ＳＵＮ－２に限れば，ＴＣＰ／ＩＰプロトコルを用いて効率の良いＴＣＣＰを実現出来る。

等の検討を行ったが，色々な制約等により実現出来なかった。

よって，現在の環境内で，かつ最小の変更で済み，また上位にＤＤＢ層を乗せられるだけのパフォーマンスを得られる様に，次の改良を行った。

- 機能の劣るＵ－1200をシステムからはずす。
- ＮＩＵのエミュレーターをＳＵＮ－２内に実現し，ＳＵＮ－２内での折り返しとする。又，ＤＬＥとＮＩＵエミュレーター間はパイプではなく，4.2ＢＳＤのプロセス間通信機能を用いる。
- 非同期入出力機能は，4.2ＢＳＤ固有のselectシステムコールを用いる。

select関数は複数の入出力対象に対して非同期入出力を行なえる関数で，指定した入出力対象の集合（ＬＩＳ通信処理プロトコルでは入力用パイプの集合となる）に入力データが存在しないか又は出力不可の場合にはプロセスはブロックされ，集合内のいずれかの対象が入出力可能になるとプロセスに制御が返り，かつ入出力対象を通知される。このselect関数の使用により，ホスト内のＣＰＵ負荷は減少した。

上記の改良により，ＴＣＣＰは格段と効率を上げ，ＤＤＢＰの実現に耐えられるものとなった。

ＤＤＢＰの開発として，ＴＣＣＰの単純群サービスを用いて分散型トランザクション処理を実現した。しかし今回実現したのは，２相コミットメント手法によるコミットメント制御を，同一ＤＢＳ（ＳＵＮ－２上のＲＤＢのgranada）上に実現したにすぎず，同時実行制御の部分は残されている。しかし，今回の目的が，分散型データベースシステムを効率的に行う為に開発された群通信を評価するという意味では十分であったと思われる。また，今回開発したＤＤＢＰ－Ｐプロセスが備えているオペレーションは，基本となる操作を提供する様になっており，たとえれば，ＤＤＢＰのマシン語に相当するものである。よって，これらのオペレーションを用いて，ＤＤＢＰの拡張及び上位の応用層の構築が容易に行なえるであろう。

ＬＩＳ通信処理プロトコルは，分散型データベースシステムの処理を効率的に行う為に開発され，通信処理の核となり群通信を提供するトランスポート群制御プロトコル（ＴＣＣＰ）と，ＴＣＣＰ上で分散型データベースシステムをサポートする分散型データベースプロトコル（ＤＤＢＰ）を実現した。今まで述べた様に両プロトコルとも実用に用いるには，まだまだ解決しなければならない問題が残されている。しかし，現在商用として実動している分散型データベースシステムが無く，又コンピュータの利用形態が集中処理から分散処理に移行している現在では，その一つの実現例として意味を持つと思われる。

4. パソコン用関係型ＤＢＭＳ

情報システムにおいて，個人用のワークステーションが主要な構成要素となってきている。こうしたワークステーション（ＬＳ）を構成する主要な機能は，データベース・システム（DBS）と，通信プロセッサ（ＣＰ）である。このために，ワークステーション，パソコン用のＤＢＳの管理システムとしてのデータベース管理システム（DBMS），PRDBMSの開発を行った。PRDBMSの概要について述べる。

PRDBMSは，開係型言語RQL（Relational Query Language）を利用者に提供する関係型のDBMSである。現在ＰＣ－９８００のMS－DOS ver. 2.0のもとで実働する。ただし，主記憶として，５１２KB以上が必要である。

PRDBMSは，図4.1に示すモヂュール構成をしている。PRDBMSは，図4.1に示すように，次の三つのインタフェースをもっている。

(1) ＲＱＬインタフェース
(2) ＴＭＬインタフェース
(3) ＶＳインタフェース

4.1 ＲＱＬインタフェース

ＲＱＬインタフェースは，利用者に，非手続的な関係型言語ＲＱＬを提供する。ＲＱＬの構文を次に与える。

＜ＲＱＬ文＞ :: ＝＜var 文＞＜get 文＞＜delete 文＞＜append 文＞
＜replace 文＞

＜get 文＞ :: ＝get［＜結果関係名＞］（＜目標リスト＞）
［＜論理式＞］

＜目標リスト＞ :: ＝＜目標式＞{，＜目標式＞}

＜目標式＞ :: ＝［［＜結果属性名＞］［／＜タイプ＞［＜長さ＞］］＝］
＜項＞

＜タイプ＞ :: ＝i（整数）| d（倍精度整数）| r（実数）|
f（倍精度実数）| c（文字）| j（日本文字）|
t（時間）

＜項＞ :: ＝＜属性変数＞|＜定数＞|＜関数＞|（＜項＞）|－＜項＞|
＜項＞＜算術演算子＞＜項＞

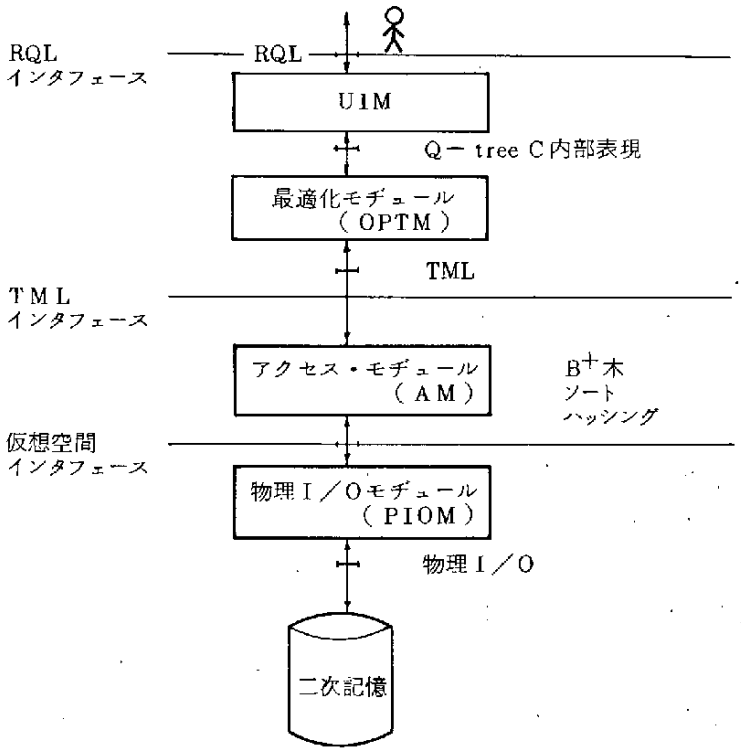


図 4.1　PRDBMSのモヂュール構成

＜算術演算子＞ :: ＝＋｜－｜＊｜／｜％｜＊＊

＜関数＞ :: ＝＜算術関数＞｜＜集約関数＞

＜算術関数＞ :: ＝＜算術関数名＞（＜項＞{，＜項＞}）

＜算術関数名＞ :: ＝sin｜cos｜tan｜exp｜ln｜log

＜集約関数＞ :: ＝＜集約関数名＞（＜項＞{，＜項＞}
{ by ＜項＞} where ＜論理式＞）

＜集約関数名＞ :: ＝count｜any｜exist｜min｜max｜sum｜
count｜aver

＜論理式＞ :: ＝＜述語＞｜（＜論理式＞）｜～＜論理式＞｜
＜論理式＞＜論理結合子＞＜論理式＞

＜論理結合子＞ :: ＝＆＆｜｜｜～

＜定数＞ :: ＝＜文字列＞｜＜数＞｜＜時間＞

＜文字列＞ :: ＝"＜文字＞{＜文字＞}"

＜数＞ :: ＝＜整数＞｜＜実数＞

＜実数＞:: ＝［＜整数＞］［.］［＜整数＞］［±［＜符号＞］＜整数＞］

＜時間＞:: ＝?［＜整数＞］［@＜整数＞］

（例　1986年2月1日12時8分58秒

?860201 @ 120858　　）

例えば，次の関係を考える。

EMP（ eno，ename，age，dno ）

DEPT（ dno，dname ）

このとき，以下の問合せは，RQLで次のように書ける。

(1) 開発部員名をもとめよ。

```
var ( e, EMP ) ( d, DEPT ) :
get ( name = e. ename )
  d. dno && d. dname = "開発"
```

(2) A氏と同じ部員名を求めよ。

```
var ( e, EMP ) ( el, EMP ) ( d, DEPT ) :
get ( el. ename )
  e. ename = "A" && e. dno = el. dno and
  e. eno ~= el. eno ;
```

(3) 部毎の平均年齢を求めよ。

```
var ( e, EMP ) ( d, DEPT )
get ( d.dname, avage / r = aver ( e.age, e.eno by d.dno
                                  where e.dno = d.dno )) :
```

(4) 会社全体の平均年齢以上の人の開発又は調査部毎の平均年齢を求めよ。

```
var ( e, EMP ) ( d, DEPT )
get ( d.dname, avage / r =
      aver ( e.age, e.eno by d.dno
        where e.dno = d.dno and
          e.age >= aver ( e.age, e.eno )))
    d.dname = "開発" | |
    d.dname = "調査" ;
```

(5) 開発部のA氏の所属を調査部にかえよ。

```
var ( d, DEPT ) ( dl, DEPT ) ( e, EMP ) ;
replace e ( dno = dl.dno )
```

e.ename＝"A" && e.dno＝d.dno &&
d.dname＝"開発" &&
dl.dname＝"調査" ;

(6) 最高齢の人の情報を消去せよ。

var (e, EMP)
delete e e.age＝max (e.age) ;

(7) 新たに，A氏と同年齢のC氏を，B氏と同じ部として加えよ。

var (e, EMP) (el, EMP) (d, DEPT)
append EMP (eno＝1061, ename＝"C",
age＝e.age , dno＝d.dno)
e.ename＝"A" && el.ename＝"B" &&
el.dno＝d.dno ;

4.2 TMLインタフェース

TML (tuple manipulation language) は，関係の組単位の操作を行うものである。以下に，その仕様を与える。

4.2.1 データ構造

図4.2に，PRDBMSにおける記憶階層を示す。仮想空間は，連続なページ (page) の連続空間で，ページには一連の番号が付けられている。現在，ページ・サイズは512 Bであるが，システム・パラメータの変更により，256 Bの整数倍の大きさにできる。

PRDBMSは，メモリ・バッファと二次記憶との二層のメモリ記憶階層をもっている。メモリ・バッファ (MB) は，ページと同じサイズのフレーム (frame) の連続領域である。

二次記憶 (SM) は，ページと同じサイズのブロックから構成されている。

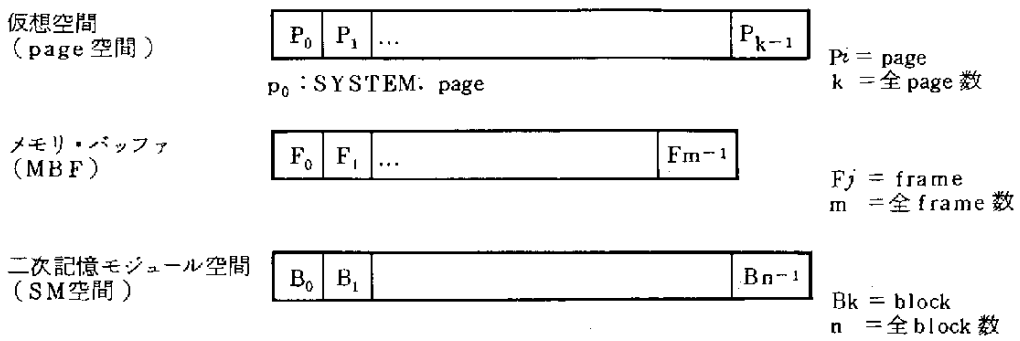
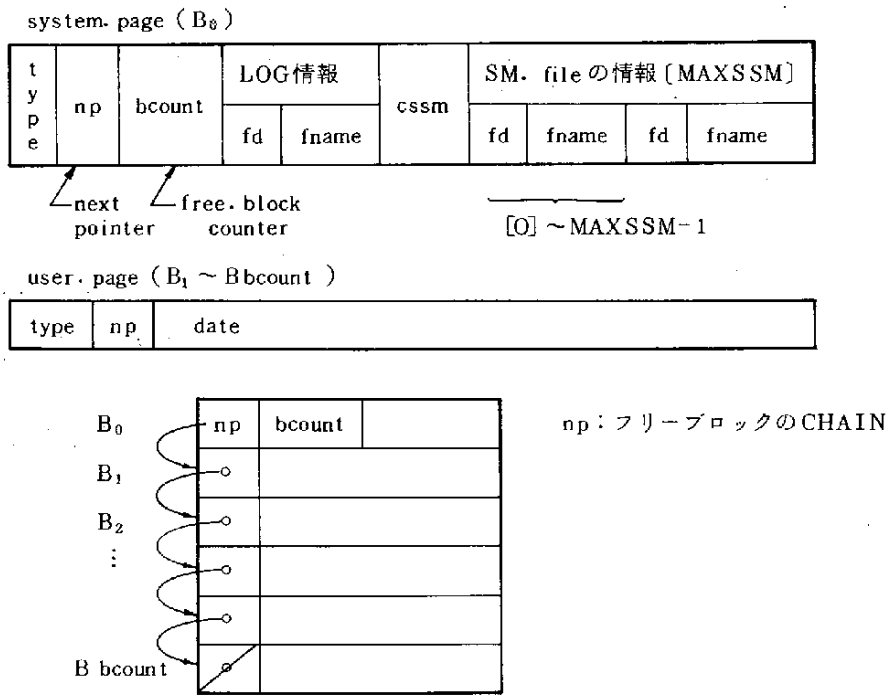


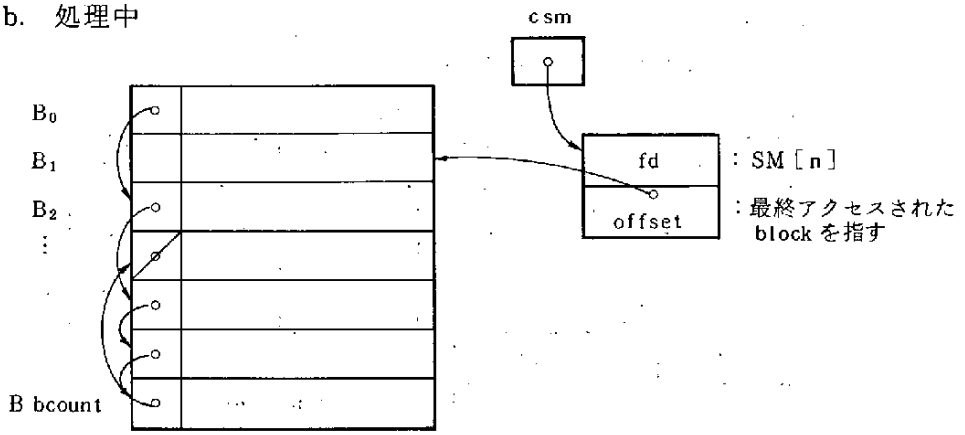
図 4. 2

(1) SMの構造

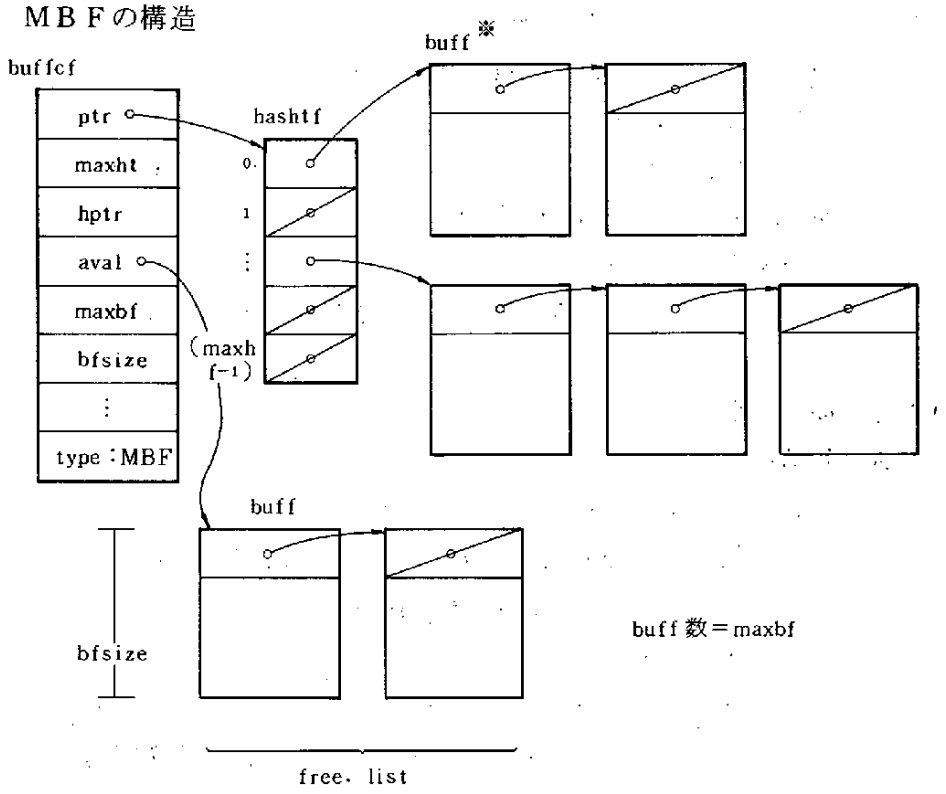
a. 初期化



b. 処理中



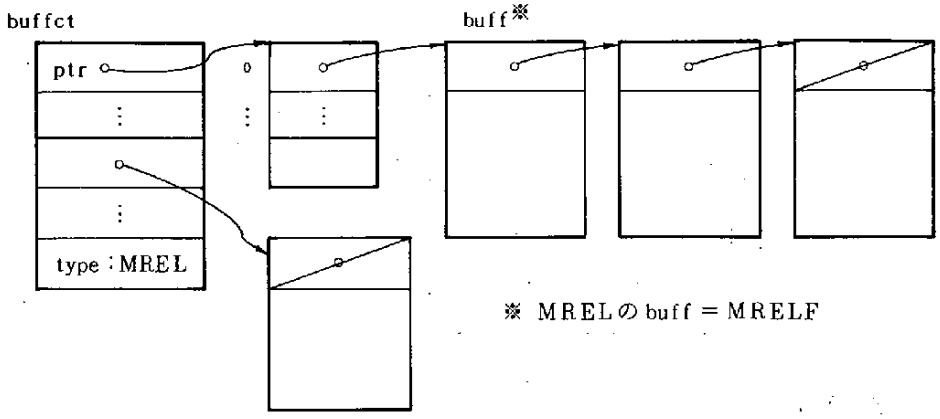
(2) ＭＢＦの構造



※ ＭＢＦの buff ＝ＦＲＡＭＥ

(3) ＤＤ／Ｄの構造

MREL　　Relation用バッファ



MATT　　Attribute用バッファ

ＭＲＥＬと同様

但し，type：MATT.　MATTのbuff＝MATTF.

MIND　　Index用バッファ

ＭＲＥＬと同様

但し，type：MIND.　MINDのbuff＝MINDF

4.2.2 操　作

A 初期化

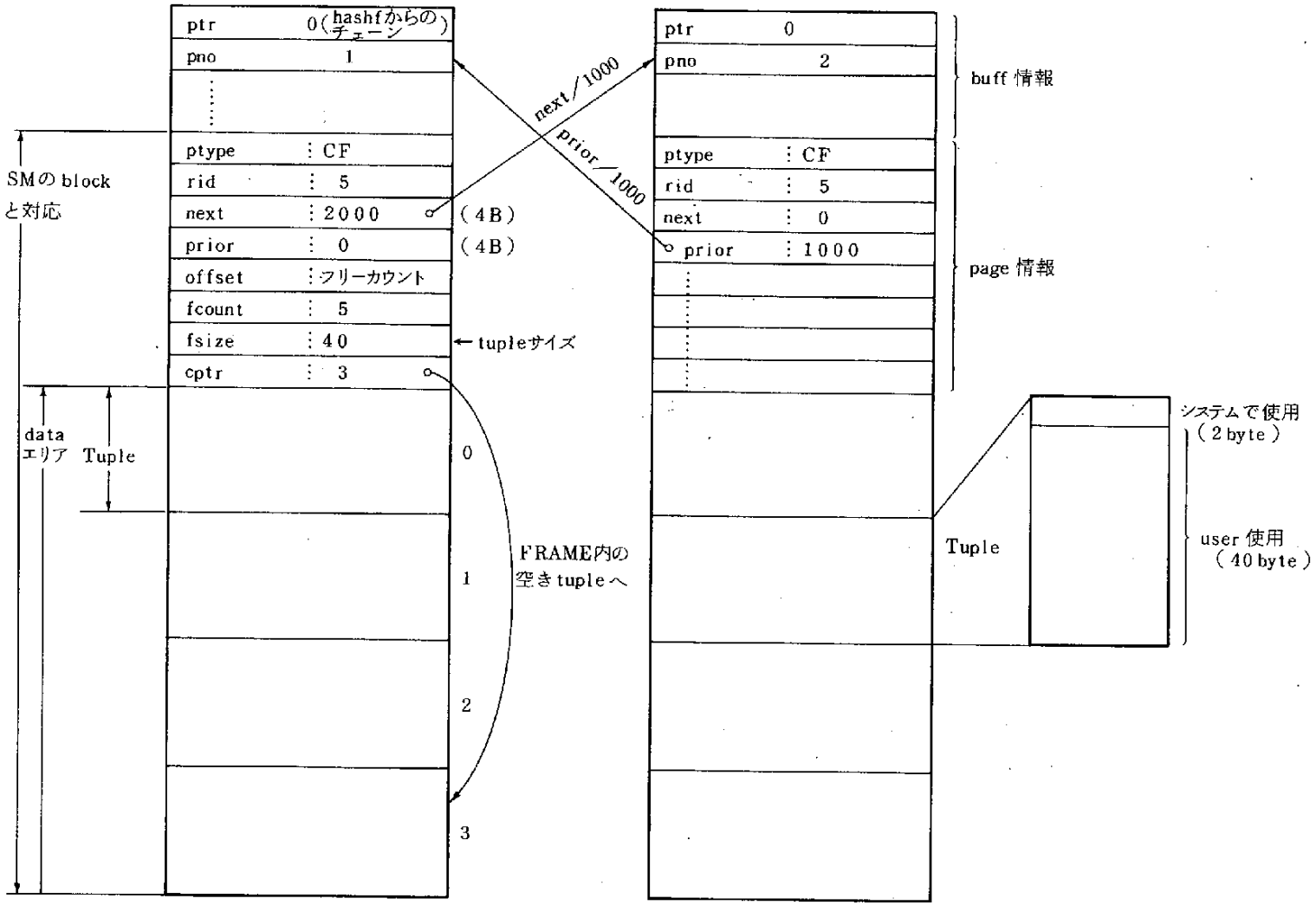
(1) Ｄ／Ｂの生成&メモリ・バッファの初期化

○＊crtdb（fname，pcnt，fcnt，hcnt，rcnt，rhcnt，acnt，ahcnt，icnt，ihcnt）

パラメータ

char	＊fname	：生成するＤ／Ｂのファイル名へのポインタ（最後に'b'を含め６文字迄とする）
PAGENO	pcnt	：Ｄ／Ｂのブロック数（MAX. 100,000）
int	fcnt	：メモリに確保するFRAME数
int	hcnt	：FRAME用hash. table数
int	rcnt	：メモリに確保するRelation用バッファ数
int	rhcnt	：Relation用hash-tableのtable数
int	acnt	：メモリに確保するAttribute用バッファ数

F R A M E


ptr 0(hashfからのチェーン)
pno 1
ptype : CF
rid : 5
next : 2000
prior : 0
offset : フリーカウント
fcount : 5
fsize : 40
cptr : 3
(4B)
(4B)
←tupleサイズ
SMのblock
と対応
data
エリア
Tuple
0
1
2
3
FRAME内の
空きtupleへ
next/1000
prior/1000
ptr 0
pno 2
ptype : CF
rid : 5
next : 0
prior : 1000
buff 情報
page 情報
Tuple
システムで使用
(2byte)
user 使用
(40byte)

int　　ahcnt　：Attribute 用 hash-table の table 数

int　　icnt　：メモリに確保する Index 用バッファ数

int　　ihcnt　：Index 用 hash-table の table 数

リターン値

FRAME　＊　：SYSTEM・PAGE（page no 0）へのポインタ

エラー　：NULL

(2) D／Bのオープン及び，メモリ・バッファの初期化

○＊opendb（fname，fcnt，hcnt，rcnt，rhcnt，acnt，ahcnt，icnt，ihcnt）

パラメータ

1－(1) crtdb と同様

リターン値

FRAME　＊　：SYSTEM-PAGE(page no 0)へのポインタ

エラー　：NULL

"MINIMUM"　1／21現在

fcnt ≧ 6

rcnt ≧ 4

acnt ≧ 41

icnt ≧ 1

B　二次記憶（ＳＭ）の拡張

(1)　ＳＭ上に free・block が無くなった時，ＳＭ上に次の File を作成する。

◦ inissm（pcnt）

パラメータ

PAGENO　pcnt　　作成する page 数

リターン値

int　inissm（　）　T. File を Createl. 初期値が設定された。

エラー　　F. File が追加できなかった。
注）

注）. SUN 使用時，拡張は，15回とするが，標準入出力，エラー出力ファイルで3ファイル＋LOGファイルが設定される時，拡張は，11回となる。

C　Relation の操作

(1)　Relation の定義

◦ crtrel（rname，degree，width，rtype，owner）

パラメータ

char　＊rname　：構築する Relation へのポインタ
（'10' を含め 17 文字迄）

short　degree　：Relation を構成する Attribute の数

short　width　：Relation を構成する Attribute のバイト数

short　rtype　：Relation の type

char　＊owner　：Relation の owner
（'10' を含め 7 文字迄）

リターン値

RELID（u. short）：Relation の ID

エラー　　：F

(2)　Attribute の定義

◦ crtatt（rid，attno，attname，type，length，offset，
comp，mode 1，mode 2）

パラメータ

RELID　rid　　：定義する Attribute の Relation - ID

short attno ：定義するAttributeのSEQUENCE・NO
（1からの連番）
char ＊attname：〃 名称
（'\0'を含め17文字迄）
short type ：〃 type
short length：〃 バイト長
（typeが文字でcomressionされる時MAX）
short offset ：〃 先頭からのオフセット値
short comp ：Compression Type（0：NO-comp.
1：SPACE）
short mode 1：0：unique. 1：non-unique
short mode 2：0：NULL値要 1：NULL値不要

リターン値
int T
エラー F

(3) Relationの検索

a. Relation名での検索

○＊nremrel（rname）

パラメータ
char ＊rname ：検索するRelation名

リターン値
MRELF＊ ：Relationのバッファへのポインタ
エラー ：NULL（指定したRelationが存在しない）

b. Relation-IDでの検索

○＊retmrel（rid）

パラメータ
RELID rid ：検索するRelationのID

リターン値
MRELF＊ ：Relationのバッファへのポインタ
エラー ：NULL（指定したRelationが存在しない）

(4) Attributeの検索

a. Attribute名での検索

○ ＊nrematt（rid，attname）

パラメータ

RELID rid ：Attributeが属するRelation－ID

char ＊attname：Attributeの名称

リターン値

MATTF ＊ ：Attributeのバッファへのポインタ

エラー ：NULL（指定のAttributeが存在しない）

b. Attribute.NOでの検索

○ ＊retmatt（rid，attno）

パラメータ

RELID rid ：Attributeが属するRelationのID

short attno：AttributeのNO

リターン値

MATTF ＊ ：Attributeバッファへのポインタ

エラー ：NULL（指定のAttributeが存在しない）

(5) Relationのスキーマ&データ削除

○ dstrel（rid）

パラメータ

RELID rid ：削除するRelation－ID

リターン値

int ：T

エラー ：F

(6) Relationのデータ削除

○ delrel（rid）

パラメータ

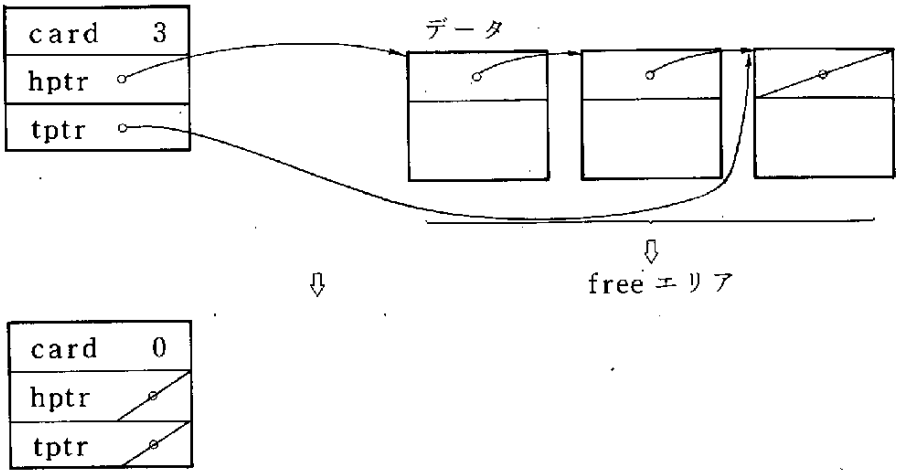
RELID rid ：データを削除するRelation－ID

リターン

int delrel（）：T

エラー ：F（rid Relationが存在しない）

rid の relation 情報



D　Relation 内の Tuple の操作

(1)　Relation へのデータ STORE

◦ tupstr (rid, S, length)

パラメータ

RELID　rid　：データを Store する Relation - ID

char　＊S　：Store するデータ

short　length：　〃　の長さ（バイト長）

リターン値

TID (u.long)　：STORE した Tuple - ID

エラー　：0　（空き Block が無い）

注）CF で Store する時，length は パラメータ数を合せる為"(short)1" 等をセット。

又，Relation の current - TID は，常に Store した Tuple - ID がセットされる。

(2)　Relation 内のデータ DELETE

◦ tupdel (rid, tid)

パラメータ

RELID　rid　：tuple が所属する Relation - ID

TID　tid　：Delete する Tuple の ID

リターン値

int　　：T

：F　（Relation内にtidのTupleが存在しない）

注）RelationのCurrentは，next Tupleが有れば，next－TID。

next－Tupleが存在しない場合NULLに置き換る。

(3) Relationの先頭Tupleサーチ

○＊first（rid，t）

パラメータ

RELID　rid　：Relation－ID

TID　＊t　：Tuple－IDを返すエリアのポインタ

リターン値

FRAME　＊first（）：Tupleが所属するFRAMEへのポインタ

エラー　：NULL

(4) Relationの最終Tupleサーチ

○＊last（rid，t）

パラメータ

RELID　rid　：Relation－ID

TID　＊t　：Tuple－IDを返すエリアのポインタ

リターン値

FRAME　＊last（）：Tupleが所属するFRAMEへのポインタ

エラー　：NULL

(5) Relationの内で直前にアクセスを行った（現在指示子が示す）Tupleの次のTupleをサーチ

○＊next（rid，t）

パラメータ

RELID　rid　：Relation－ID

TID　＊t　：Tuple－IDを返すエリアのポインタ

リターン値

FRAME　＊next（）：Tupleが所属するFRAMEのポインタ

エラー　：NULL

(6) Relationの内で，直前にアクセスを行ったTupleをサーチ

◦ ＊curren（rid，t）

パラメータ

RELID　rid　：Relation－ID

TID　＊t　：Tuple－IDを返すエリアのポインタ

リターン値

FRAME　＊curren（）：Tupleが所属するFRAMEのポインタ

エラー　：NULL

(7) Relation内で直前にアクセスを行ったTupleの1つ前のTupleをサーチ

◦ ＊prior（rid，t）

パラメータ

RELID　rid　：Relation－ID

TID　＊t　：Tuple－IDを返すエリアのポインタ

リターン値

FRAME　＊prior（）：Tupleが所属するFRAMEのポインタ

エラー　：NULL

(8) Relationの内から，指定されたTuple－IDのTupleをサーチ

◦ ＊find（rid，t）

パラメータ

RELID　rid　：Relation－ID

TID　＊t　：Tuple－IDのセットされたエリアのポインタ

リターン値

FRAME　＊find（）：Tuple－IDが所属するエリアのポインタ

エラー　：NULL

(9) Relation内で直前にアクセスを行った（現在指示子が示す）Tuple－IDを更新する。

◦ point（rid，tid）

パラメータ

RELID　rid　：Relation－ID

TID　tid　：更新するTuple－ID

リターン値

int　　point( ) : T (正常)

エラー　　: F (Tuple - IDがRelationに存在しない)

⑩ Relation内で直前にアクセスを行った(現在指示子が示す) Tuple - IDを求める。

◦ accept (rid)

パラメータ

RELID　　rid　: Relation - ID

リターン値

TID　accept( ) : Tuple - ID

エラー　　: NULL (Tupleが存在しない)

⑪ Tupleの長さを求める。

◦ tuplen (tid)

パラメータ

TID　　tid　: Tuple - ID

リターン値

Short　　: Tupleの length (バイト)

: F　エラー

⑫ Tupleの取り出し

◦ tupload (tid, str)

パラメータ

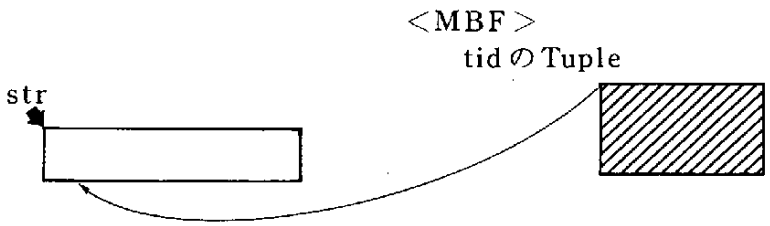
TID　　tid　: Tuple - ID

char　＊str　: Tupleを loadするエリアのポインタ

リターン値

Short　tupload( ) : loadしたTupleのサイズ (バイト)

エラー　　: F (tidのTupleが存在しない)



⑿ Tuple内，１項目の取り出し

◦ attload（tid，rid，attno，str）

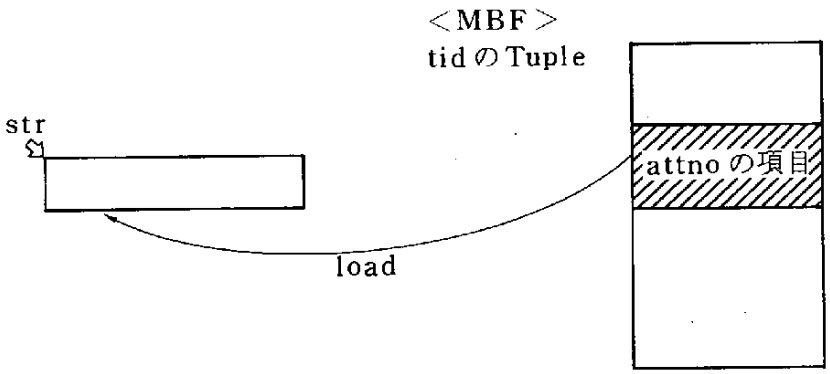
パラメータ

TID	tid	：Tuple－ID
RELID	rid	：Relation－ID
short	attno	：項目NO.
char ＊	str	：取り出した項目をloadするエリアのポインタ

リターン値

short attload（）：loadした項目の長さ

エラー　：F（tidのTupleが存在しない
又は，attnoの項目が存在しない）



⒀ 項目の長さ計算

◦ attlen（rid，attno，tid）

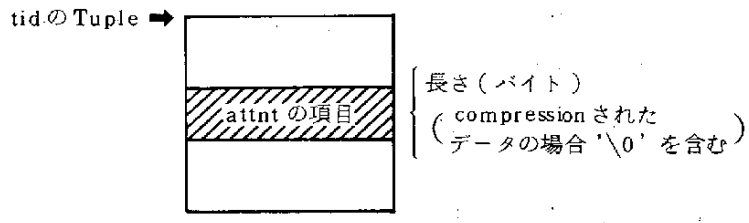
パラメータ

RELID	rid	：Relation－ID
short	attno	：項目NO.
TID	tid	：Tuple－ID

リターン値

short attlen（）：項目長（バイト）

エラー　：F（Relation－ID，Tuple－ID
attnoのエラー）



E　インデクス

関係の任意の属性に，インデクスをつけている。インデクスとしては，B^+木を用いている。

(1)　Relationの，1つのAttributeにindexを持たせる。

◦ Crtind（rid，attno）

パラメータ

RELID	rid	：indexを付けるRelation－ID
ATTNO	attno	：〃　Attribute－NO

リターン値

int	T	：

エラー

int	F	：

(2)　Tuple中にIndex付きAttributeがあれば，indexテーブルへTuple－IDを登録する。

◦ luindex（rid，tid）

パラメータ

RELID	rid	：Tupleの所属するRelation－ID
TID	tid	：登録したいTuple－ID

リターン値

int	T	：

エラー　　F　：

(3)　indexの取り消し

◦ delind（rid，attno）

パラメータ

RELID	rid	：取り消しを行うRelation－ID

ATTNO　attno：取り消しを行うAttribute－NO

リターン値

int　T　：

エラー　F　：

(4) Tupleの全index属性の取り消しを行う。

◦ delitid（rid, tid）

パラメータ

RELID　rid　：Tupleの所属するRelation－ID

TID　tid　：Tuple－ID

リターン値

int　T　：

エラー　F　：

(5) Tupleの1Attributeに対して，indexの取り消しを行う。

◦ deliat（rid, attno, tid）

パラメータ

RELID　rid　：TupleのRelation－ID

ATTNO　attno：Tuple中，取り消しを行うAttribute. NO.

TID　tid　：Tuple－ID

リターン値

int　T　：

エラー　F　：

(6) indexのサーチ（keyでサーチ）

◦ findkey（rid, attno, v, vlen, tid）

パラメータ

RELID　rid　：サーチするRelation－ID

ATTNO　attno：　〃　Attribute－NO

char　＊v　：　〃　文字例へのポインタ

short　vlen　：　〃　〃　の長さ（バイト）

∴－1の時は，attribute. length

TID　＊tid　：リターンする，Tuple－IDへのポインタ

リターン値

int	T	：同一のkeyがサーチできた
	＊tid	：〃　　を持つTuple－ID
〃	F	：（＊tidが／＝NULLの時，指定したkeyの次の値を持つTuple－IDをリターン
		（＊tid　＝＝NULLの時 keyがみつからない

(7)　findkeyで位置付けされた，Tuple－IDの次のTuple－IDをサーチ。

◦ nextkey（rid，attno）

パラメータ

RELID	rid	：Relation－ID
ATTNO	attno	：Attribute－NO

リターン値

TID	：／＝NULL
エラー	：NULL（ATEND）

F　コミットとアボート

PRDBMSでは，1つのRQL問合せをトランザクションとして管理している。

(1)　メモリ・バッファ（MBF）の内容を全て二次記憶へ書き込む。

◦ vcommit（　）

リターン値

int	T	：
エラー	F	：{S／M / LOG}のI／Oエラー

・vcommit迄のLOGは，取り消され新しくLOGが生成される。

(2)　メモリ・バッファ（MBF）の内容を全て二次記憶へ書き込む。

◦ fwrtall（　）

リターン値

int	T	：
エラー	F	：S／MのI／Oエラー

・LOGは継続される。

F 終　了

(1) D／Bの Close

◦ Closedb（　）

リターン値

int　　：T

　　：F

・MBF上のデータは全てSMにコピーされる。

(2) D／Bの Delete（SM, LOGを全て消去）

◦ dstdb（　）

リターン値

int　　dstdb（）：T　正常

エラー　　：F（I／Oエラー）

(3) D／BをLOGの内容で1処理単位前に戻す。※

◦ vabort（lfd）

パラメータ

int　　lfd　　：LOG・ファイルの File・descripter

リターン値

int　　vabort（）：

エラー　　：F（メモリが確保できない，

　　LOGが生成できない）

・vabort 迄のLOGは，Deleteされ新しく，LOGが生成される。

※1処理単位：open（create）D／B　～ vabort，

　　〃　　～ vcommit 等。

5. 高度利用者インタフェース

5.1 エキスパート・システム

5.1.1 エキスパート・システム概略

近年，人工知能研究の一分野であり，人間（専門家）の有する知識を計算機に格納し，種々の問題解決を行うためのソフトウェア・システムを構築することを目指している、知識工学に対する期待が高まっている。この理由の一つとして，知識工学に基づくシステムが，初期の研究において非常に優れた問題解決能力を示したことがある。また，近年第5世代コンピュータプロジェクト（ICOT）を初めとし，計算機メーカ等が人工知能研究を行っており，いくつかの分野においては，それを応用したシステムが報告されている〔CACM 85，JYOHO 85〕。

知識型システムにおいて要求される事項は，まず，人間（専門家）と同程度の問題解決能力を有するということである。このためには，問題解決に必要となる，ある対象分野における大量の知識と呼ばれる情報を格能する必要がある。この知識情報は，数値を主に取り扱う従来の情報を包含し，かつより広範囲な情報（記号データ，数式データ等）である。知識情報処理では従来のデータ処理に比べ複雑かつ大量のデータの処理が必要となるが，近年のVLSI技術の発展により大規模計算機の発展に負うところが多大である。さらに，その技術的背景としては，データベース・システム技術，記号処理言語による実現技術，プログラミング言語の進化等がある。

知識工学的技術に基づくシステムは，一般に知識型システムと呼ばれている。こう呼ばれる理由は，問題解決において人間の有する知識を切り出し計算機において知識情報を利用しているからである。特にある専門分野（例えば，医療診断，設計問題等）に問題を絞り，その分野の専門家の協力を得ながら，問題解決を行うシステムを，エキスパート・システムと呼ぶ。これは，知識型システムの一種である。エキスパート・システムにおいては，重要な問題として，人間をサポートすることを目的とし，そのための利用者インタフェースとなる機能，言語等を整理，充実することが必要である。この時，エキスパート・システムの利用者は，計算機に対する非専門家ということになる。したがって，先の利用者インターフェースが非専門家にも容易なものであることが重要である。この非専門家のコンサルテーションに主眼をおいて開発されたエキ

スパート・システムが，エキスパート・コンサルテーション・システムである。現在までに開発されたエキスパート・システムの多くは，この類に属するシステムである。この概略を図5.1に示す。

エキスパート・システムを設計する場合には大きく次の三種の重要な課題があると考えられている〔FEIGE77〕。

- 知識の表現
- 知識の利用
- 知識の獲得

これらについては，議論が多々あるので，他文献，例えば，〔UENOH85〕〔JYOHO85〕）を参照されたい。

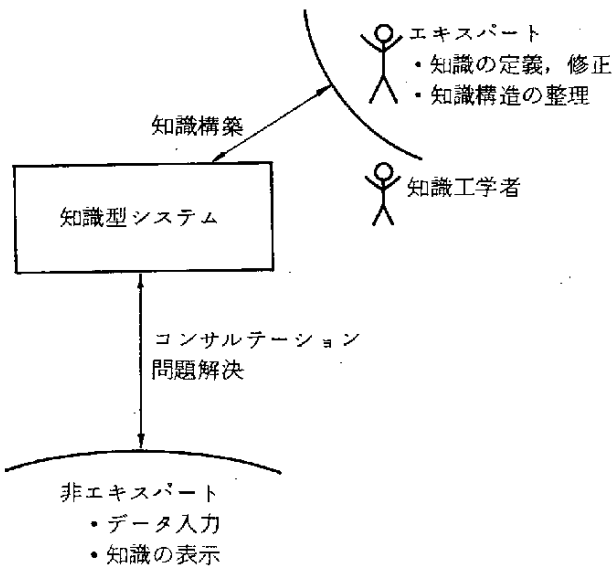


図5.1　エキスパート・システムの概略

ここでは，実用システムとしての知識型エキスパート・システムをどのような点が，実際に構築，設計を進める上で重要な課題となるかについて述べる。

(1) 知識の表現

ある専門分野における知識をどのように表現し，知識と呼ばれる情報をいかに計算機に実現するかということである。これは，知識型システム構築のための1つの鍵となるものである。なぜなら，この表現を中心に知識獲得および知識利用が考えられるからである。主たる表現形式としては，フレーム，プログクション・ルール，セマンティク・ネットワーク，および論理に基づくものがある〔 IEEE 83 〕。後述するＬＩＰにおける表現は，論理型表現形式である。これらの知識表現モデルに対する利点，欠点等については〔 UENO 85 〕に述べられている。

(2) 利用者インタフェースの問題

知識型エキスパート・システムは，一種の人間ー機械系を構成している。人間（専門家，非専門家）との柔軟性のある対話を行うために，知識型システムに専門家の有する知識をある表現形式に構成し，これを格納することが必要である。また，この蓄えられた知識を用いて非専門家に対しコンサルテーションを行う。このため，もし利用者インタフェースの設計思想および利用者ーシステムのインタラクション等が混乱したものとなっているならば，利用者（専門家，非専門家）に対しシステムの動作および格納された知識の体系が不透明なシステムとなる。一般的に，知識ベース・システムに対する操作は，知識型システムが処理することが可能である言語，コマンドおよびメニュー画面等を用いて操作される。これらの言語は，知識型システムの利用者インタフェースとなり，知識獲得支援またはコンサルテーションを行う。この利用者インタフェースにより，システムと人間のインタラクションが行なわれる際，その機能としては，次のようなことが行なわれる。

- 知識の定義，更新
- 推論結果の表現および推論の説明　　etc.

ここで，推論とは，知識型システムにより行なわれる問題解決のためのプロセスである。

(3) 大規模知識データの管理

一般に，知識型システムにおいて取り扱われる世界は，開いた世界の一部である。この一部をシステムとして構築する。したがって，知識として格納される情報は，種々雑多な性質（例えば症状，故障原因，物理的デー

タ等）が入力される。これを管理し知識の無矛盾性や一貫性の管理は非常に困難な問題点を提起している。その原因は以下のようなものであると考えられる。

(i) 情報が主として定性的である。

例えば，「手足がむくんでいる」というものである。これは，コンサルテーションを受ける非専門家および知識提供者たる専門家の主観および経験を通じ得られたヒューリスティクスと呼ばれる知識が多々ある。エキスパート・システムが，要求された問題解決能力を得た理由の一つには，このヒューリスティクスな知識が中心的役割を果たしたということであった〔FEIGE 77〕。この種の知識に対して，定性的な評価，検証等が一般には困難であろう。

(ii) 知識を独立に説明することが困難である。

先に述べたように，現段階において開発されたエキスパート・システムは，医学診療診断システムに代表されるような（e.g.,MYCIN〔SHORE 76〕，PIP〔SZOLP 78〕，INTERNIST〔SZOLP 78〕）比較的あいまいな分野を対象として開発されている。これらのシステムにおいては，医者（専門家）の有する知識とは大きく異なる構造を有する知識表現形式が用いられていることもある。近年においては，医者や設計エキスパートの有するその分野の構造に関する知識，公理および法則やその関連知識等を表現することも試みられている〔例えば，DAVIR 83〕。また，推論過程および知識の整合性を保つためのシステムの開発が徐々に進められている。例えば，論理的な性質を明らかにするという立場に立った非単調論理〔MCDED80〕，デフォルト推論〔RITER 80〕，および第1階述語論理に基づきモデル記述のための言語体系MLL（Multi Layered Logic）〔OHSUS 85a〕等である。また，データ・ベースに対するデータ・ベース管理システムに相当する知識ベース管理システムの開発も進められている。

(4) 否定情報の処理

一般に，データベース・システムおよび知識ベース・システムにおいて否定的情報を取り扱うためにいろいろな制約が付されている。例えば，データ・ベースにおいては，現在有しているデータ以外のものは全て否定と考えるという閉世界仮説〔RITER 78〕である。一方，医療コンサルテー

ション・エキスパート・システムにおける推論においては，ある病気仮説を確定するために各種所見の入力が要求される。したがって，ある病気仮説のための条件を満たしていないということを調べるためには，非常に多大な情報が要求されることになる。

5.1.2　エキスパート・システムのアーキテクチャ

エキスパート・システムは，機能的には，「人間（専門家）と同等の問題解決能力を有し，診断や設計のために非専門家を支援するためのソフトウェア・システムである」と考えることができる。一方，アーキテクチャの立場から，一般には知識と呼ばれる情報を蓄えた知識ベース（knowledge base）と，その知識を用いた問題解決を行うための推論機構から成り立っていると，定義することができるであろう。ここで知識は，推論を行うために必要である情報（すなわち，事実および手続き）であると考えることができる（図 5.2）。実際的なエキスパート・システムでは，図 5.2 に示すような基本的構成要素の他に，知識ベース・エディタが必要となる（図 5.3）。知識ベース・エディタは，知識型（エキスパート）システムにおける知識ベース

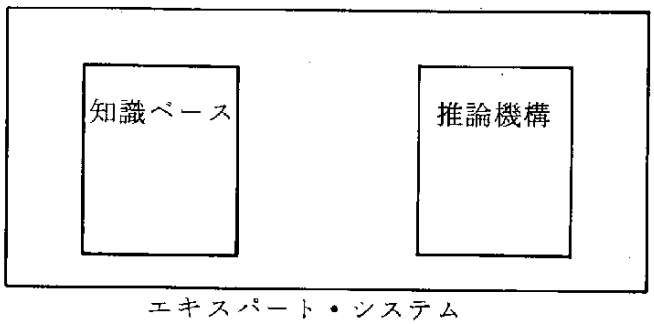


図 5.2　エキスパート・システムのアーキテクチャ

の定義，更新等を行うための構成要素の一つである。これは，一種の利用者インタフェースとなっている。このエディタにより，知識ベースは構築される。エディタの構造は，知識型システムが提供する知識表現モデルに従属している。また，この種のエディタは知識ベースに知識を構築しようとする利用者が知識ベース・エディタの提供する知識表現形式に自然な流れをもって知識ベー

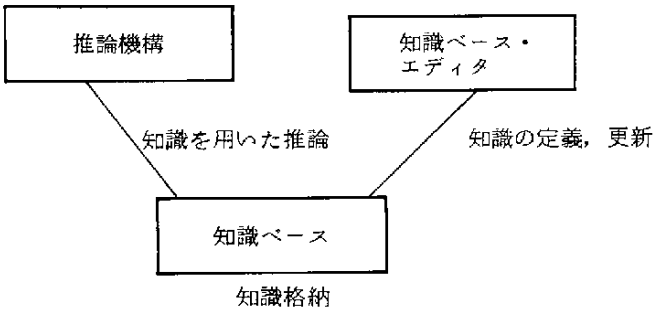


図 5.3　エキスパート・システムのアーキテクチャ

スを構築することができる機能群および設計思想を有していなければならない。さらに，知識ベース・エディタは，ある種の知識ベース管理システムの機能を果たしていなければならない。なぜなら，（知識ベースを構築しようとする）利用者が知識ベースの構成，構造等について十分な理解があると考えられない場合があることは，十分に予想される。また，知識ベースにおける表現形式のための文法が明確になっていないということ，および知識表現に対する制約条件に対する理解ができていないということに対しては利用者にそれを示し，制約条件を破壊しているような知識の定義を避けなければならないと考えられるからである。

これらの他に，知識ベース・エディタが有しなければならない要件は，知識獲得を支援するような機能を有しているということである。知識獲得の問題は，未だ十分に研究がなされておらず実用システムに対し適応可能であるということは言い難いと考えられる。したがって，利用者がスムーズに知識を定義することができるような機能が備わっている，もしくは容易にその種の機能を追加することができるという柔軟な構造を，知識ベース・エディタが有していることが望まれる。

以上の議論より，知識ベース・エディタには，次のような機能が基本的に備わっているか，もしくは容易に変更，追加することのできる構造を有していることが必要である。

(i)　知識ベースの構築のための専用エディタであること。
(ii)　知識ベース管理システムの一部の機能を有していること。
(iii)　知識獲得支援モジュールを備えていること。　　etc.

次に，実際的なエキスパート・システムの構成の例を挙げ，実用システムにおいてはどのような機能群から成り立っているかについて述べる（図 5.4）。

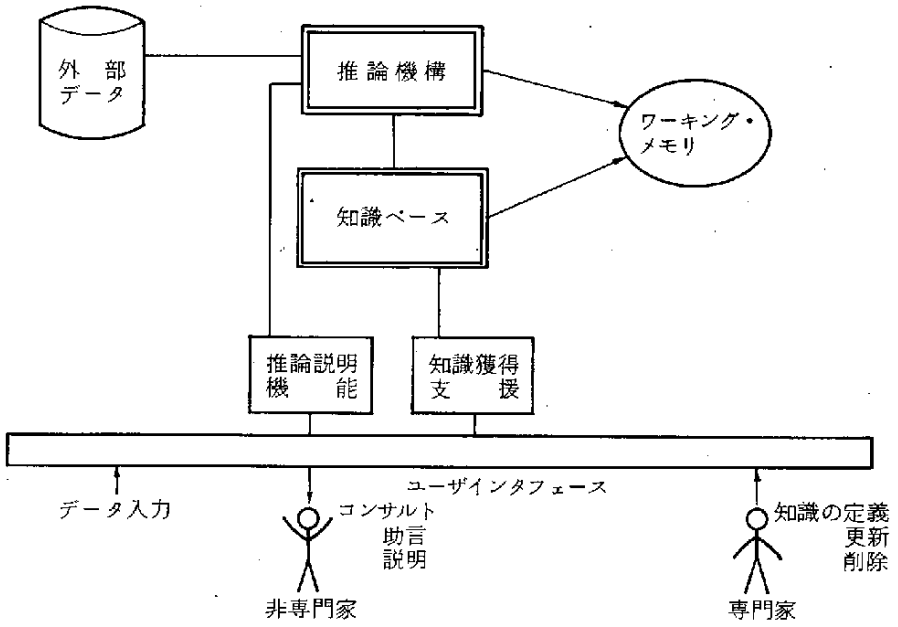


図 5.4　知識型（エキスパート）システムの構造の例

知識ベースは，ある一定の表現形式を有する知識を格納している。知識表現形式とそこに定義された知識をいかに利用することである推論制御メカニズムを合わせて知識表現モデルと呼ぶ。これは，問題解決支援を行うための知識表現モデルである。一方，この推論メカニズムの設計および実現において，利用者が独自に問題に適合したモデルとすることも可能であるようなシステムがある。これは，汎用知識表現言語であり，この種の言語については，後に述べる。知識ベースでは，知識の表現単位をどのような形式にするのか，全体として体系的にどのようにまとめるのか，メタ知識と呼ばれる知識をどのように埋めこむのか，通常の知識と同列に取り扱われるものなのか，あいまい推論を実現するためにどのようにそれに関する情報を定義，設定するのか，等ということについて具体的に定義，設定するのかということが問題となる。ここで，メタ知識とは，知識の表現単位，体系および推論自身のための知識である。例えば，推論方法を決定するための知識は，メタ知識の一種である。

推論機構は，知識ベースにおいて定義された知識群に対して，解釈を与え，結論を導くためのメカニズムである。もし必要となるならば，知識型（エキスパート）システム利用者に対して，データ入力を促したり，推論過程を提示するためのメカニズムが付加されている。ここで，推論とは，問題解決のための解決過程である。

知識型（エキスパート）システムでは，推論機構と知識ベースという基本的構成が分離されている。推論プログラムから，そのデータである知識を独立させることにより，知識の管理，知識の一部を差し換え可能なことによるシステムの柔軟性，知識ベースの拡張可能なことにより拡張性および問題分野に応じた知識ベースが構築可能であることによる汎用性，等を達成することができる。ただし，知識表現形式は推論制御メカニズム（推論機構）が構造的に分離されていることと，両者が互いに独立に存在することとは異なっている。両者は，互いに密接に関連しており，知識表現形式が決定されると，その利用方法である推論制御メカニズムの動作，および推論方法は固定されたものとなる。逆に，推論機構の構造が決まっているならば，知識表現形式も固定されたものとなる。

ワーキング・メモリは，結論を導くために必要であった中間的な結論および結果を一時的に格納するための作業領域である。一般に，ある結論を得るためには，推論機構は中間仮説と呼ばれる仮説を多々生成する。また，推論プログラムでは，中間結果を生成するであろう。このような，中間仮説，中間結果等を格納するために，ワーキング・メモリが必要である。ただし，このワーキングメモリを全く必要としないようなシステムもある〔UENOH 85〕。ここでは，知識ベースが，中間状態を保存しているからである。

基本的には，知識ベース，推論機構およびもし必要ならばワーキング・メモリがあれば，知識型（エキスパート）システムは構成されるが，実用システムとして有用となるためには，各種説明モジュール，先に述べた知識ベース・エディタ等が必要である。

外部データは，推論を進める過程において入力されたデータ（例えば，病気に関する問診，病療またはVLSIにおける構造データ，チップに関する情報）のような事実に関する知識が主体となる。このようなデータは，事実に関する知識として知識ベース内に置かれているようなシステムも多々ある。しかしながら，実行例として多くのデータの保存または大量データを用いるよ

うな問題に対しては，外部データを利用したいというような要求が起こるであろうことは十分に予想される。このような要求に応じるためには，外部にデータを保存することが必要である。このような目的に用いられるデータ・ベースの規模や構造は，目的およびシステムの構造により異なる。現段階において開発されたシステムは，研究実験システムであったということもあり，この外部データ・ベースを必要としなかったり，小規模な専用システムであったとしても，妥当であると考えることができる。しかし，今後，ＣＡＤ，ＯＡのような大規模データを取り扱い，実用システムの開発を進めるならば，大量の情報の管理が必要であろう。また，すでに，問題分野における情報が既存データ・ベースとして構築されているならば，これを利用した知識型（エキスパート）システムの構築が予想される。このような理由から，データ・ベースの利用は，十分に考えられる。

後に述べるＬＩＰは，このような要求が将来生じるであろうという観点から開発されている。これは，ＬＩＰ開発の動機の一つである。

5.1.3　エキスパート・システムとデータ・ベース

ここでは，エキスパート・システムにおける知識ベースとデータ・ベースの関係について述べる。

5.1.3.1　エキスパート・システムとデータ・ベース・システム

知識ベースは，知識の管理機構を有し，データ・ベース管理システムは，データを管理するためのシステムである。また，知識ベースは，知識と呼ばれる情報を管理，格納するためのツールである。情報管理機構および情報表現ツールとして知識ベースを考えた場合，両者は非常に似かよったものとなっている。共通となる問題点としては，次のようなものを挙げることができよう。

(i)　情報の一貫性管理
(ii)　データ・モデル
(iii)　推論システム　　etc.

以下これらについて述べる。

情報の一貫性（integrety）の管理は，知識ベース，データ・ベース両者において，重要な問題点の一つである。データ・ベースは，一般にデ

ータ・モデル（後述），基本オペレーションおよび一貫性制約により規定されてる〔DATEC 81〕。この一貫性制約の表現に対して，述語論理の枠組の中でこれを取り扱おうとするものがある〔GALLH 84〕。また，知識ベースでは，知識の一貫性および無矛盾性（ consistency ）の管理のための推論システムを構築している。例えば，新しいデータが入力されたり知識が定義された場合，従来存在する知識と比較することにそれと矛盾を生じるならば，その対象となっているデータおよび知識の入力を行なわないとするようなシステムの開発である。

データ・モデル研究は，データ・ベース研究において情報（ここでは，データ）とその上の関係に注目し，いかに情報を表現するのかということである，例えば，IFO〔ABITS 84〕は，セマンティクネットワーク〔BRACR 79〕の定式化を行う場合には，非常に有効であると思われる。さらに，データ・モデルの研究においては，あるデータをいかに計算機上に自然な形かつ効率のよい方法により表現するかということが問題となっている。これは，知識の表現という問題と深くかかわっている。ただし，知識ベース技術ではその表現技術に適合した推論手続（すなわち，推論機構設計および手続き型知識（ procedural knowledge ））の記述形式が問題となるのに対して，データ・モデルでは推論というよりむしろ基本オペレーション（e.g.検索，更新等）が問題となる。このような，手続きの位置付けが，両者では異なる場合が多い。

推論システムでは，知識データまたはデータという対象に対する考え方に相異があると考えることができる。データ・ベースに格納される情報（データ）は，客観的な事実であるのに対し，知識ベースでは経験則が重要な役割を果たした。したがって，推論システムにおいても，データ・ベースでは検索が中心となるのに対して，知識ベースではその検索の結果得られた情報に対する解釈が問題となる。

近年，このような問題点を中心に考察を進めることのできるデータ・ベース・システムが論理プログラミングの発展に伴い構築または検討されるようになった。代表的には，論理データ・ベースがそのようなデータ・ベースである。ここでは，第一階論理（述語論理）に基づき，ある種の演繹システムと組み合わされ，論理データ・ベース・システムを形成している〔GALLH 84〕。

後述するＬＩＰにおけるデータ・ベースでは，CODASYL〔CODAS 73〕モデルに対し一種の論理的なインタフェースを設けることにより，論理データ・ベースとしている。

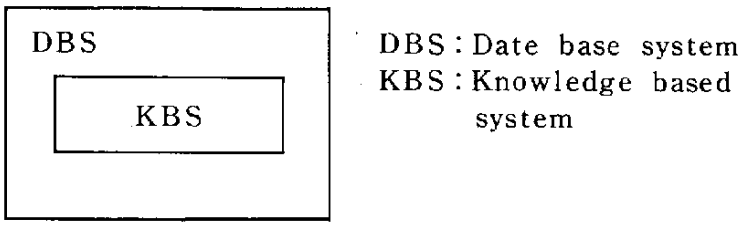
5.1.3.2　データ・ベースと知識ベースの結合

情報管理システムという観点からは，知識ベースとデータ・ベースは多々の共通点がある。さらに，問題解決において，どのようにデータ・ベース知識ベース技術を用いるかが，目的によりその利用形態はさまざまであろうが，これからの情報システムを考えてゆく上で重要な問題の１つであるということが言われている〔UENOH 85，OHSUS 85 b〕。

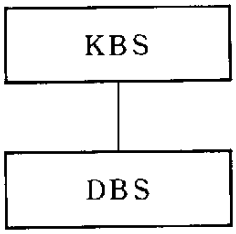
データ・ベース技術は，大量の（客観的）事実の管理を行なおうとするものであると考えることができる。例えば，データ・ベースでは，時間が変わることにより事実の一部が変更されるが，そのデータ・ベースが有する正規な構造というものは不変である。かつ，この技術における重要な概念は，データ独立（date independence），格納されているデータの一貫性（integrity）および矛盾性（consistency），制約（constraints）である〔DATEC 81〕。

一方，知識ベース技術においては，問題領域に含まれる経験的知識が重要な役割を果たす。この種の経験的情報は，正規な表現と呼ばれるような形式はおそらくはないであろう。また，知識ベース技術において重要なことは，表現能力の増強と推論メカニズムにおける操作の容易性であろう。

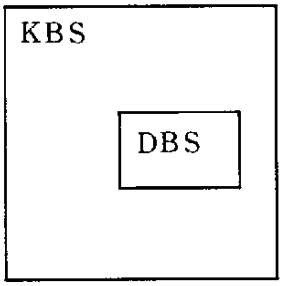
最近，知識ベース技術とデータ・ベース技術結合に関する試みが進められている。その結合の立場には次の三種があると考えられる（図5.5）。



(a)　データ・ベース・システムの機能として



(b) データ・ベースと知識ベースが同等として



(c) 知識ベースの一部として

図 5.5 知識型システムとデータ・ベースシステムの結合

(a) データ・ベース・システムの機能の一部として知識ベースが存在する。

(b) データ・ベース，知識ベースが共存している。

(c) 知識ベース・システムの機能の一部としてデータ・ベース技術が用いられている。

それぞれについて簡単に述べる，これは知識ベース技術とデータ・ベース技術をどのように整合性を図りながら，それぞれの技術的立場より他システムを考えるのかという問題と深くかかわっていると考えられる。

(a)の方法は，データ・ベース・システムに対し知識ベース技術により得られた成果を利用し，データ・ベースに対する知的なアクセス，セキュリティ，一貫性および無矛盾性のチェック等の機能を実現する，というような方法である。

(b)の結合方法は，ＫＢＳおよびＤＢＳを独立なシステムであると見なし，ＫＢＳが取り扱う問題解決のためにデータ・ベースに格納されている情報

を用いるというようなことである。この方法では，大規模データについてはＤＢＳに，リアルタイムな処理が要求されるルール等についてはＫＢＳ側に，格納することが可能である。ここで，ＫＢ側およびＤＢ側に格納される情報の性質は，その情報の有する性質および目的による。

(c)の方法は，DBSをKBSの機能の一部として含んでしまう方法である。この方法では，KBSは知識管理システムであるとともにデータ・ベース管理システムとなっている。また，これにはDBS技術をKBS技術の中に含むような構造もあると考えられる。すなわち，利用者は特に知識ベースに格納されている情報およびデータ・ベースに格納されている情報として区別する必要はない。(b)の方法との相違点は，(b)ではKBS，DBSに格納されている情報を完全に区別しているという点である。したがって，(b)に基づくシステム利用者は，その情報の問い合わせに関して，ＤＢＳのための命令体系を記述しなければならない。しかしながら，(c)の方法では特に区別する必要はなく，利用者は同一の命令体系によりシステムに対する操作を行うことが可能である。図5.6において，現在提案されている，知識管理システムの概略を示す。この図に示すシステム（KMSと呼ばれている。）

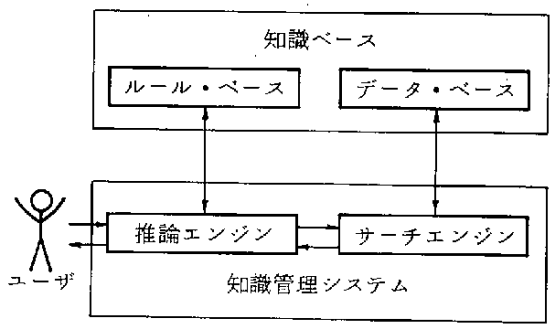


図５－６　知識管理システムの概略〔KELLC 86〕

では，サーチエンジンとしてリレーショナルデータ・ベース・システム，知識ベースとしてはルール・ベースをそれぞれ想定している。ここで，ルールというのは，プログラション・ルールではなく，論理プログラミング言語におけるルールである。

5.1.3.3　階層型データ構造の定式化

本セクションにおいては，階層型データ構造とその論理的な記述について述べる。ここで階層型データ構造を定式化する理由は次の点のようなことである。

(i) 階層型データ構造は，知識表現モデルとしてよく利用されている。このとき，階層構造におけるノードは，a－kind－of（AKO）と呼ばれるリンクにより結合されている。このようなシステムの例には，ZERO〔ITOH 86〕等，一連のフレーム型データ構造がある。

(ii) 階層型データ構造より記述された知識ベースに対する知識体系計算メカニズムを解明する。

(iii) 人工知能研究において用いられるデータ構造の数学的な性質の解明。

(iv) 人工知能研究とデータ・ベース研究（特に，論理データ・ベース）の技術的な接点として，その可能性を評価する。

(v) CODASYLモデルに対する定式化と階層型データ構造の性質を比較し明らかにする。

(vi) フレーム・モデルのある制限された場合における論理的な知識表現能力の評価。この場合，特に注目するのは，階層の上位において定義されたそのノードの属性が一部または全て下位ノードのものに受け継がれるというインフリタンスの機能を述語論理を用い記述するということ，である。

〔1〕 アプローチの方法

階層型データ構造を定式化するにあたり，いくつかの用語を非形式的に述べる。まず，階層型データ構造はフレーム（frame）と呼ばれるデータ構造〔MINSM 75〕をノードとし，各フレームの構成要素をスロット（slot）と呼ぶことにする。知識ベースは，このフレームをノードとする階層構造であり，各フレームはある概念対象（conceptual objects）を記述している。この時，スロットは，その概念対象の属性，性質等を記入している。各フレームを結合するリンクは，AKOリンクと呼ばれ，全体の構造はインフリタンスのメカニズムの有効に利用するということを前提とする階層構造となる。

ここでは，図5.7に示すようなアプローチをとることにする。現段階において，我々が注目すべき点は，次の2つのことである。

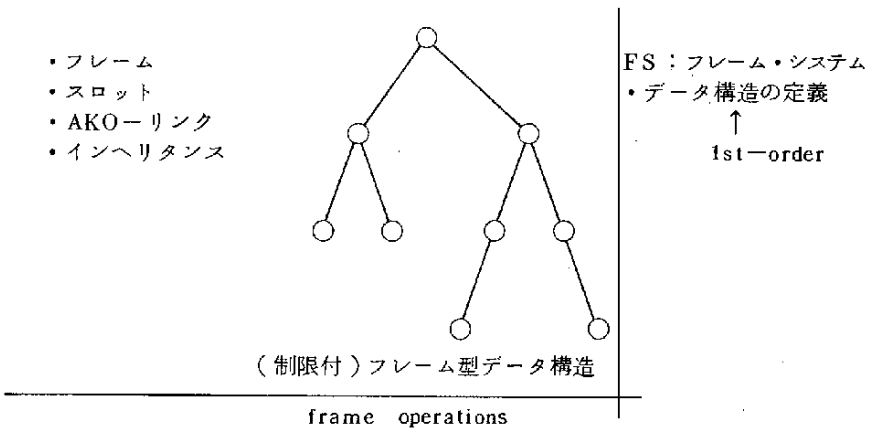


図 5.7　アプローチの概略

(i)　階層型データ構造の定式化。この時，データ構記述のために第一階言語を用いる。

(ii)　フレーム型データ構造の再構成。フレームモデルに対して基本演算を定義する。

上記２点に対して，有効なる定式化を行う。

〔2〕　フレーム・モデル

現在までに開発されているフレーム・モデルの特徴はデータ構造が汎用性の高い階層型データ構造であること，メッセージ交換による推論メカニズムの実現およびポインタを用いたフレーム相互の参照関係の記述等である。今回は特に，その階層型データ構造に注目した定式化を行う。

従来の汎用フレーム・モデル〔e.g. KL－ONE〔BRACR 85〕U-nits〔STEFM 79〕，ZERO）上に論理型言語インタフェースを設けめるためには，フレーム・モデルの第１階論理に基づく言語体系により，そのデータ構造を再定義する必要がある。フレーム・モデルでは，物理と論理の両面が混在している（特にZEROでは）ので，論理的な整理が必要である。このためにまず，フレーム・モデル・スキーマ　£＝（$，Ø，ℂ）を定義し，データ構造である$を基に，フレーム・システムを定義する。フレーム・システムは第１階理論を用い定義される。表5.1に定式化のための記号表を示す。以降この

表に基づき定式化を進める。

表 5.1　記号のまとめ

フレーム・システム（データ構造）の定義：FS

- 木構造　G＝（N, R, T, FT）
- フレーム構造

  frame　　𝔽＝（F, F）

  relationship　　ℝ＝（R, R）

- フレームデータスキーマ
- フレームデータインスタンス

  S＝（𝔽, ℝ）

  フレーム・データ構造　𝕊＝（S, S）

- 基本的オペレーション＝𝕆
- 制約条件（constraints）＝ℂ
- フレーム・モデルスキーマ：L＝（𝕊, 𝕆, ℂ）
- フレーム・システム⇦第1階理論

  FS＝＜S, Ls, As, Is＞

  S：フレーム・データ構造

定式化の戦略は，フレーム・スキーマを定義するために，木構造（フレーム木），フレーム，スロットに対しデータ構造を規定する。次に，基本 operation 𝕆，および制約条件ℂを定義し，フレーム・モデルの構造を明らかにする。これが，フレーム・システムＦＳである。

〔3〕 フレーム木（Frame tree：G）

フレーム木Gは，フレームをノードとする木構造であり，各ノードはa－kind－ofと呼ばれるリンクにより結合されている。ノードである各フレームは，ある概念対象を記述している。GはG＝＜N, R, T, FT＞により定義される。

〔定義〕 フレーム木：G

G＝＜N, R, T, FT＞

Nは，フレームの集合であり，N＝C∪Iである。ここで，Cはクラスフレーム（class frames），Iはインスタンスフレーム（in-

stnce frames）をそれぞれ示す。RはR⊆N×Nであり，ノード間の関係を示す。T＝{class，instance}であり，IT＝N→Tなる写像である。

ここで，class，instanceは，ZEROが有するフレーム型（frame type）である。図5・8にフレームの一般的構造を示す。フレーム型 instanceはある問題領域における例示を，またclass は抽象的概念を，それぞれのフレームが記述していることを示している。

Frame－name	Frame－type
A－Kind－Of slot	
D. Descendants slot	
Description－information slot	
CreatedBy slot	
ModifiedBy slot	
Slot 1	
Slot 2	
:	
Slot n	

Frame－type ： instance
subclass

図5.8　フレームの構造

〔4〕 データ構造

ZEROにおけるデータ構造は，図5.9に示すデータ構造に制限し考える。主たる制限は以下のものである。

(1) システムがフレーム型データ構造を管理するためのスロット（e.g., D. descendents slot, Created by slot 等）については現段階においては考えない。

(2) スロット－値というペアを基にする。図5－9(a)は，フレーム型データ構造を簡単化したものであり，(b)は基となる概念的データ構造を示している。(a)においては，上記制限より，システムがフレー

ム型データ構造を管理するためのスロットFrame－type，AKOスロットのみを有している。ＡＫＯスロットには，その値として階層の上位フレームのポインタが記入されている。(b)は次のようなことを表わしている。まず，フレームは，一種のスキーマ（ schama ）であり，その構成要素であるスロットに値が設定されたものはそのスキーマの実現値（ instance ）である。スキーマ，実現値は，それ

F

Frame－name	Frame－type
AKO	F_1
A	a
B	b
C	c
D	d

(a)

id	Frame-name	Frame type	AKO	A	B	C	D	← schema
n	F	Cor I	F_1	a	b	c	d	← instance

↑自然数

(b)

図 5.9　制限付フレーム構造

ぞれ frame schema $\underline{\mathbb{F}}$, frame instance $\mathbb{F}$と呼ぶことにする。スキーマ化されているフレームには，フレーム－idを有している。このidは自然数であり，フレーム木Gに対して右廻りに番号が付されているものとする。

ここでは，フレームにおける構造である，フレーム相互の関係について定義する。この，フレーム上の関係を，Frame Relationshipと呼ぶことにする。

〔定義〕　Frame：$\mathbb{F}=(\underline{F}, F)$

$\underline{F}$：a frame schema

$\underline{F}=\{@F\}\cup\Omega_F$

Ω_F：a set of slots

@F：frame－id

これは，自然数からフレーム木Gに対して右廻りに番号が付されている。

F：a frame instace

$F \underline{C}$ dom($\underline{F}$)

ここで，@Fは，フレーム固有の識別子となっている。dom($\underline{F}$)は，$\underline{F}$のドメインを示している。

現在，考えられる限定されたデータ構造においては，フレーム上の関係は，ＡＫＯと呼ばれるリンクにより結合されている，ということのみである。したがって，関係$\mathbb{R}$のタイプとしてはＡＫＯのみを考えることになる。

〔定義〕　Relation $\mathbb{R}$

$\mathbb{R}=(\underline{R}, R)$

$\underline{R}$：relation schema

R：relation instance

$\underline{R}=(\underline{F_1}, \underline{F_2})$

$\underline{F_1}$＝parent frame schema

$\underline{F_2}$＝child frame schema

$R \subseteq$ dom(F_1, F_2)

関係$\mathbb{R}$は，relation schemaとこのinstanceのペアにより定義される。この階層構造の関係は，図5.10のように示される。

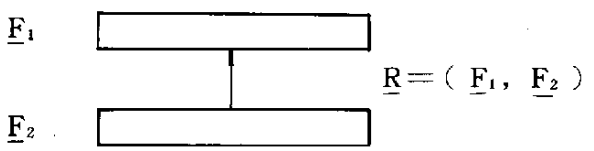


図５－10　フレームの階層構造

このように，$\mathbb{F}$を定義したが，スロットの値のドメインは次のようになる。frame schema $\underline{F}$は，項目集合｛@F，t_1……，tm｝（$m \geq 0$）で与えられる。@Fはframe－id，t_iはスロットである。各スロットsに対して，dom(s)は，sのドメインである。Frame $\underline{F}$に実際のデータを与えたframe instanceの集合Fの各要素について，Fは

$$F \subseteq \mathrm{dom}(@F) \overset{m}{\underset{i=1}{\times}} \mathrm{dom}(t_i)$$

である。

次に，フレーム型データ構造を定義する。フレーム木のノードである$\mathbb{F}$，関係$\mathbb{R}$については上に定義した。フレーム型データ構造は，この二者により定義される。

〔定義〕 frame date structure：$

$\$ = (\underline{S}, S)$

$\underline{S}$：frame date schema

$\underline{S}$：frame date instance

〔定義〕 frame date schema

$\underline{S} = (\underline{\mathbb{F}}, \underline{\mathbb{R}})$

$\underline{\mathbb{F}}$＝a set of frame schema

$\mathbb{R}$＝a set of frame schema

$\underline{\mathbb{F}}$，$\underline{\mathbb{R}}$は，frameおよびrelation schemaの集合である。ここで，それぞれの集合は有限である。

〔5〕 基本オペレーション

次に，抽象的な仮想的なフレーム操作関数を設ける。これは基本演算（operation）Ⓞとなる。ここでKBは知識ベースを，frame，slotはそれぞれフレーム，スロットを表現している。

基本演算としては，次のものがある。

(1) Frames（KB）

現在注目しているフレーム型データ構造において定義されているフレーム名を求める。

(2) Slots（Frame）

Frameにより示されるフレーム内において定義されているスロッ

トを求める。

基本演算(1)，(2)はそれぞれ知識ベースとなるフレーム型データ構造の対象となるドメインを求めるような関数となっている。

(3) Slotval (slot frame)

スロット slot，フレーム frame において示されるスロット値を求める。

(4) Current－frame ( )

フレームに対する現在子を返す。ここで，現在子は，フレーム木上の巡行のために設定されている。この現在子をフレーム現在子と呼ぶことにする。

(5) Right－frame ( )

フレーム現在子において指されているフレームの兄弟フレームにフレーム現在子を設定する。兄弟フレームがないならばＮＩＬを返す。サーチの順序は，right－to－left の規則に従う。

(6) Child

フレーム現在子を，そのフレームの子フレームに移す。子フレームがないならばＮＩＬとなる。サーチの順序は，depth－first 規則に従う。

(7) Current－frame－slot ( )

スロットに対する現在子を返す。ここで現在子は，フレーム木上の巡行のための表示子である。Current－frame では，フレームそのものに注目していたが，この関数は最初に First－slot を動作させることにより有効となる。すなわち，あるフレームに定義されているスロットについて，そのスロットと同じ名前を有するスロットを含んでいるフレームを巡航により求める。このための現在子をスロット現在子と呼ぶことにする。

(8) First－slot (slot)

slotにより示されるスロットを含むフレームを返す。この時，スロット現在子は，発見されたフレームとなる。もし，このようなフレームが発見されないならばＮＩＬとなる。

(9) Next－slot ( )

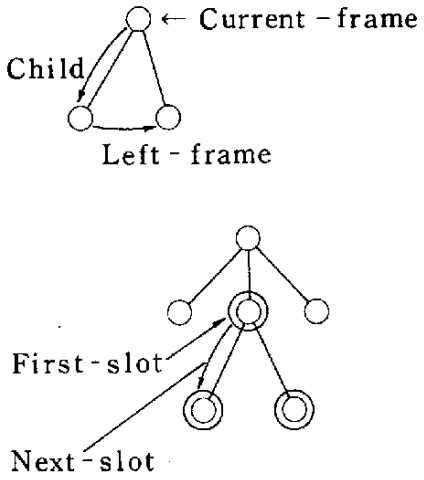
関数(8)により設定されたスロット現在子を，条件を満たすフレー

ムに設定する。このとき巡行規則は，フレーム木において，depth－first right－to－left の規則に従う。

このように，9種の基本操作関数を定義する。その概略を図5.11に示す。ここではフレームの更新やメッセージ交換については，まだ導入されていない。これにより，フレームスキーマに対する基本演算が定義された。例えば，論理式（S，＊F，Inst）において，S，＊F，Instが，それぞれスロット，フレーム（ただし変数），Instとなっている場合，これを満たす＊F（フレーム）を発見するプログラムは，図5.12のようになる。

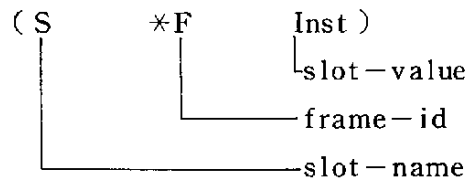
operations：⓪

(1) Frames（KB）
(2) Slots（frame）
(3) Slotval（slot，frame）
(4) Current－frame（　）
(5) Left－frame（　）
(6) Child（　）
(7) Current－frame－slot（　）
(8) First－slot（slot）
(9) Next－slot（　）



㊟・サーチの順序……depth－first right－to－left
・検索のみ

図5.11　基本演算群



*Fの値を求める。

「あるフレームにおける slot Sの値は Inst である」

```
N=( COND ( NULL ( CURRENT-SLOT S ) ) NIL ) )
   ( COND ( EQUAL ( SLOTVAL S
                          ( CURRENT-FRAME-SLOT ) )
                  INST ) )
          ( CURRENT-FRAME-SLOT ) )
   ( NEXT-SLOT S )
   ( GO TO N )
```

図 5.1 2　基本演算利用例

〔6〕 制約条件

フレームスキーマ：£：($, O, ℂ) における制約条件には次のものがある。

(1) $R_i = (F_{i1}, F_{i2}) \in \mathbb{R}$　各 i について

ただし，$F_{i1}, F_{i2} \in F$

(2) フレーム木にはサイクルがない。

(3) AKOについて

フレーム木における親は唯一存在する。

$R_i (F_{i1}, F_{i2}) \wedge R_j (F_{j1}, F_{j2}) \wedge F_{i2} = F_{j2} \rightarrow F_{i1} = F_{j2}$

(4) instance frame は child frame を有しない

$R_i = (F_{i1}, F_{i2})$ について

$R_i (F_{i1}, F_{i2}) \rightarrow FT(F_{i2}) \neq$ instance

以上である。

ここまでの議論，定義において，フレーム・モデル・スキーマ£＝($, 𝕆, ℂ) が定義された。

〔7〕 フレーム・システム

ここでは，フレーム・システム：FSを定義する。これは，第1階

理論に基づいた形式的体系である。ＦＳは，フレーム・データ・構造Ｓ，１階言語L_s，公理A_sおよび推論規則I_sより成り立っている。この体系を図5.13に示す。フレーム・システムは，図5.13に示した4つ組により，定義される。

言語L_sは，アルファベットの集合よび，wffから成り，アルファベットには種々の述語記号より成る。フレーム，およびスロットは，それぞれm項（$m>0$）述語および二項述語として定義されている。この様（図5.13参照）な記号集合に対して，項とwffs（well－formued formulas）は，〔ENDEH 72〕と同様に定義することができる。

$FS=<S, L_s, A_s, I_s>$

S : frame data structure

L_s : a first－order language for $\underline{S}(=(A, W))$

A : alphabets $(=C_s \cup V_s \cup P_s \cup L_c)$

W : a set of wffs.

C_s : a set of constants, c_1, c_2 ……

V_s : a set of variables, v_1, v_2 ……

P_s : a set of predicates $=S_s \cup F_s \cup R_s \cup N_s$

S_s : a set of slot predicate symbols, S_1^2, S_2^2, ……

F_s : a set of frame predicate symbols, F_1^n, F_2^n, ……

R_s : a set of relation predicate symbols, R_1^2, R_2^2, ……

N_s : a set of normal predicate symbols,

L_c : $\wedge$, $\vee$, $\sim$, $\longleftrightarrow$, $\forall$, $\exists$, (,)

A_s : a set of axioms

I_s : a set of inference rules.

図5.13　ＦＳの基本的構成

言語L_sの解釈は，ドメインと写像の集合から成る。これをMとすると，Mは各定数aに対してM(a)はドメイン内の要素を，各m項述語記号P_sに対してはM(P) $\underline{c}$ D^nを与える。ここで，フレーム述語記号F

に対して，M(F)は frame instance 集合を，各関係Rに対しては relation instance 集合を対応づけることにする。

次に公理集合 A_s について考える，公理としては，論理的公理と等式公理に加え，次の特殊公理がある。

(1) あらゆる frame schema について

$\underline{F}=(k, x_1, x_2, \cdots, x_n)$

$\rightarrow @F(k) \wedge s_1(k, x_1) \wedge s_2(k, x_2) \wedge \cdots \wedge s_n(k, x_n)$

ただし，@F(k)は frame－id として k をとる。

(2) $F(k, x_1, \cdots, x_n) \wedge F(k, y_1, \cdots, y_n)$

$\rightarrow x_1 = y_1 \wedge x_2 = y_2 \wedge \cdots \wedge x_2 : y_n$

次に，relation schema について述べる。

(3) $Rj = (F_1 j, F_2 j)$ について

ここで $f \in F$（f＝frame occurence）であるとすると

$(f_1, f_2) \in R$

さらに，

$R(f_1, f_2) \rightarrow F_1(f_1) \wedge F_2(f_2)$

(4) $\underline{R}$ の type は AKO である。この場合には，次の公理が成立する。

(i) $R(f_1, f_2) \wedge R(f_3, f_2) \rightarrow f_1 = f_3$

(ii) $R(f_1, f_2) \rightarrow \Omega(f_1) = \Omega(f_2)$

ただし，$\Omega = \Omega_{F_1} \wedge \Omega_{F_2}$

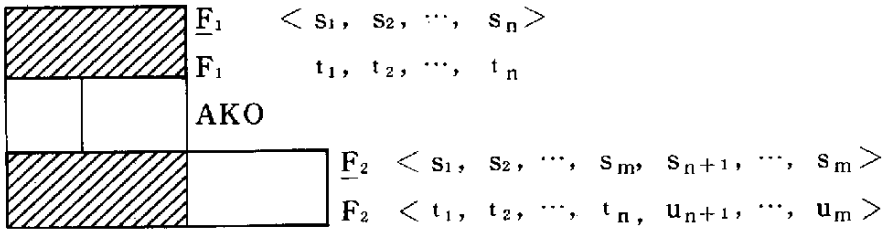
(ii)の性質は，property inheritance と呼ばれる機能である。これは，階層の上位において定義された属性が，下位に対し受け継がれてゆくという性質である。ここでは，例外処理が1つの問題点となるが，これは例外全てを列挙するという方法により避けることができる。この概略を図5.14に示す。

推論規則 I_s は，modus ponens と一般化規則を有しているとする。

この結果次のような命題を得ることができる。

〔命題〕 FSは完全で健全である。

〔証明〕 第1階理論であるので明らか。



〔例外処理〕

$$\forall t_i \forall u_j \quad F_2(t_1, \cdots, t_n, u_1, \cdots, u_m)$$
$$\wedge (\quad F \neq \text{例外} - 1\ ) \quad \left.\begin{array}{l}\\ \\ \end{array}\right\} \text{例外は全て列挙}$$
$$\wedge (\quad F \neq \text{例外} - 2\ )$$
$$\vdots$$
$$\rightarrow F_1(t_1, \cdots, t_n)$$

図5.14　インヘリタンスの公理化

5.2　論理型インタフェース：LIP

5.2.1　はじめに

最近における計算機利用技術では，種々の分野，例えば，OA，CAD 等においてデータベースシステム（DBS）が用いられている。また，一方では近年における知識型システム（KBS）および論理データベースシステム（LDB）の発展に伴い大量の情報管理メカニズムという観点からDBSとのと結合が図られるようになってきた〔UENOH 85，OHSUH 85b〕。この様にDBSの適応分野が広がるにしたがい，理論的側面からは関係型〔CODDE70〕が用いられる一方高性能性が要求される大規模DBSに対してはCODASYL型DBSが用いられている。また，DBSにおいては，従来の定型的な利用に加えて，新たに非定型的な利用に対応してめく必要がある。

LIP（Logical Interface Processor）は，従来のCODASYL DBS〔OLLET 78〕上の非手続き的インタフェースとなる言語を処理するプロセッサである。ここで，CODASYL DBSは高性能性が要求される分野には適していると考えられているため，LIP では，関係型よりもCOAASYL型DBSに対して，非手続き的言語によるアクセスが可能となるシステムの実現を目指すものとする。既にCODASYL DBS上の非手続き的インタフェースとしては，LDP〔TAKIM 83〕および〔DAYAU 82〕等が開発さ

れている。この種の非手続き的インタフェースにおいては，ソフトウェアの生産性の向上という点から，DMLを用いた場合に比べて優れているという評価が得られている〔TAKIM 83〕。LIPにより処理される言語は，第一階述語論理に基づく言語体系であるProlog〔KOWAR 79，LLOYD 84〕である。このような言語は，視野を非決定的，再帰的に定義できる点が従来のSQL〔DATEC81〕等にない機能である。これは，論理データベースおよび関係型DBSにおいて，試みられている〔CHANC81，LID84〕。

LIP研究の目的は，以下のようなことである。

(i) CODASYL型DBSに対する論理型インタフェースの提供。

(ii) CODASYL型DBSの論理的整理。

(iii) CODASYL型DBSの有する特徴を生かした導出（resolution）〔ROBIJ 65〕の改良。

以上のことを目的とする。CODASYL型DBSを用いる理由は，高性能DBを実現できること，データベースが構造情報を有していることである。

本報告書においては，ＬＩＰの設計の背景および実現の一部について報告する。ＬＩＰにおける上記目的達成のための理論的背景については，詳細に検討された。尚，本報告書の内容は，文献〔TAKIM 85〕に従うものとする。

5.2.2 CODASYLモデル

従来のCODASYLモデルは，物理的な側面と論理的な面が混在している。したがって，論理型インタフェースを設けるためには，この物理，論理両面を明確に分離する必要がある。ここでは，CODASYLモデルの論理表現を与えるために，CODASYL型のデータ構造を抽象化した概念網型データ構造と，この論理表現を与える。次に，そのデータ構造に対して抽象的な操作演算を定義する。

5.2.2.1 概念網型データ構造

従来のCODASYLデータ構造は，集合と，集合間の部分関係から構成されると考えられる。これが，概念網型データ構造となる。このデータ構造は，時間不変な論理構造を与えるスキーマと，これに実際のデータを与えた時間可変なデータ・ベースで与えられる。スキーマには，レコード(R)型と親子(S)型の二種からなる。

R型Aは，項目集合｛ A, t_1, …, t_m ｝($m \geq 0$)で与えられる。 Aはdb鍵， t_i はデータ項目($i = 1, \cdots, m$)である τ に対して，dom(τ)は， τ の定義域である。R型Aに実際のデータを与えたレコード実現値(RO)集合Aは，

$$A \subseteq \mathrm{dom}(A) \times_{i=1,\cdots,m} \mathrm{dom}(t_i) \cdots (1)$$

である。Aの各要素aをR実現値とし，aの項目 τ の値を τ(a)と示す。A内の各R実現値のdb鍵 Aの値は異なっている。

二つのR型BからAに部分関数が存在しているとき，親子(S)型C＝(A, B)が定義される。Aを親，Bを子とし，図5.15の様に表す。S型Cに実際にデータを与えたものは親子実現値(SO)集合 $C \subseteq A \times B$ で，部分関数 $C : B \rightarrow A$ を示している。Cの各要素＜a，b＞をS実現値とする。

以上のRO集合とSO集合の族が概念網型データベース(CDB)である。CDB内のR実現値aに対して，(a，b)又は(b，a)なるS実現値が存在するとき，bはaに隣接するという。いま，aから出発して，隣接R実現値を，次々に巡りながらbに到達したとき，この経路をaとbの間の路という。aから自分自身への路を巡回路という。CDBが巡回路を含むとき，CDBは巡回的であるという。

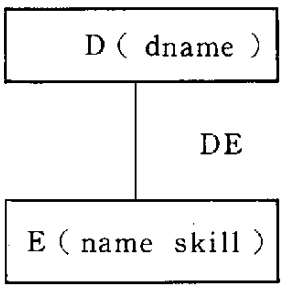


図 5.15　概念網型データ構造の例

5.2.3　概念網型体系

5.2.2において定義されたデータ構造に対して，第1階論理に基づく形式的体系 $CNS = < S, L_s, P_s, \triangle_s >$ を定義する。

〔定義〕　概念網型体系CNS

$CNS = < S, L_s, P_s, \triangle_s >$

S　：スキーマ

L_s：概念網型言語（CNL）

P_s：公理集合

$\triangle_s$：推論規則

ここで，CNLを構成する記号集合A_sは，次の4種の述語記号を有している。R，S，TおよびV述語記号である。インフォーマルには，R記号，S記号はそれぞれレコード型およびセット型を表わしている。T述語記号は，項目の定義域を指示する。それ以外の述語はV述語記号である。

スキーマSは，〔JACOB 82〕と同様に文脈自由文法を用いて，三つ組＜T，N，RL＞で表す。Tは終端記号集合である。Tは，鍵記号集合Kとデータ項目記号集合Iとの和である。Nは，非終端記号で，R記号集合RRとS記号集合SSの和である。RLは，生成規則で，次の二種がある。

$LR = \{ A = (@A, i_1, \cdots, i_m) : A \in RR, @A \in K, i_i \in I \}$

$LS = \{ C = (A, B) : C \in SS, A, B \in RR, A \neq B \}$

スキーマSが与えられているとき，概念網型言語（CNL）L_sは，記号集合A_s，論理式（wff）集合W_sの対で与えられる第一階言語である。記号集合はA_sは，定数，変数，関数記号，述語記号，論理記号（$\wedge$，$\vee$，$\sim$，$\rightarrow$，$\longleftrightarrow$，$=$，$\exists$，$\forall$）から成る。述語記号には，一項のT述語記号，n項のR述語記号，二項のS述語記号がある。スキーマSの各項目記号tに対して，T述語記号tが，各R記号Aに対して，R述語記号Aが，各S記号Cに対して，S述語記号Cが述語記号として与えられる。

この様な記号集合A_sに対して，項とwffは，〔ENDEH 72〕と同様にして定義される。

言語L_sの解釈I_sは＜D，M＞であり，Dは定義域で，Mは次の様な記号集合からの写像である。Mは，各定数aに対して，M(a)はD内の要素を，各n項関数記号fに対して，D上の関数$M(f): D^n \rightarrow D$を，各n項述語記号Pに対して，D上の関係$M(P) \subseteq D^n$を与える。ここで，各R述語記号Aに対して，M(A)はRO集合を指示し，各S述語記号Cに対して，M(C)はSO集合を指示し，各T述語記号tは，項目tの定数域dom(t)を指示する。

次に，公理集合P_sを考える。公理としては，論理的公理と等式公理に加

えて，次の特殊公理がある。

(1) 各R述語記号A＝(∃A, i_1, …, i_m)に対して

$$@k, x_1, \cdots, x_m (A(k, x_1, \cdots, x_m) \rightarrow @A(K) \wedge i_1(x_1) \wedge \cdots \wedge i_m(x_m))$$

(2) 各R述語記号A＝(@A, i_1, …, i_m)に対して，

$$\forall k, x_1, \cdots, x_m \; \forall y_1, \cdots, y_m \; (A(k, x_1, \cdots, x_m) \wedge A(k, y_1, \cdots, y_m) \rightarrow \wedge_{i=1, \cdots, m} x_i = y_i).$$

(3) 各S述語記号C＝(A, B)に対して，

$$\forall k, h (C(k, h) \rightarrow @A(k) \wedge @B(h)).$$

(4) 各S述語記号C＝(A, B)に対して，

$$\forall k, h \; \exists x, y \; (C(k, h) \rightarrow A(k, x) \wedge B(h, y)).$$

(5) 各S述語記号C＝(A, B)に対して，

$$\forall k, h, l (C(k, h) \wedge C(l, h) \rightarrow k = l).$$

以上の特殊公理を充足する解釈が，CNSのモデルである。推論規則$\triangle_s$としては，分離規則と一般化規則がある。〔KOBAI 85〕は，各種データ・モデルに対して，非常に一般的な公理を定義している。特殊公理を充足する解釈は，CNSのモデルとなる。wff wについて，⊢wはwは定理であるということを表現している。CNSは，次のような性質を有している。

〔命題〕 CNSは，完全で健全である。

〔証明〕 CNSは，第一階理論なので明らかである。□

5.2.4 仮想網型機械

今定めた概念網型データ構造に，順序関係を加えた仮想データ構造Vを定め，DMLに対応したVの操作を行う仮想網型機械(VNM)を考える。さらにVNM上において動作する仮想操作演算(VOP)を定める。VOPは，DML〔OLLET 78〕における操作演算に相当している。

5.2.4.1 仮想網型データ・ベース

仮想網型データベースは(VDB)は，仮想RO(VRO)集合Xと仮想SO(VSO)集合Zの族である。概念網型データ構造のRO集合Aに対し

て，VRO集合Xは，Aをdb鍵の大小関係で全順序化した集合である。Xの要素をVR実現値という。

S型C＝（A，B）のSO集合Cに対するVSO集合Zは，

$Z = \{ < a, m(a) > : a \in A \quad \wedge m(a) \neq \phi \}$

ここで，m(a)は，AのR実現値aの子bの集合を，ある順序関係$<_z$により全順序付けた集合である。$<_z$としては，Bのある項目についての大小関係と，任意の順序とがある。Zの要素をVS実現値という。

5.2.4.2　仮想網型操作演算

抽象的な仮想網型機械（VNM）を設け，この動作により，従来のDMLの意味を明らかにする。VNMは，レジスタとして，α_1，…，α_n，…，τ，δ_0，各VRO集合X毎のδ_Xとβ_X，各VSO集合Z毎のδ_Zを持つ。ここで，δ_0，δ_Xおよびδ_Zは，現在子〔OLLET 78〕に相当している。仮想操作演算（VOP）と，これに対するVNMの動作を以下に示す。ここで，Zの親と子を各々XとYとする。

〔VOP〕

(1)　first(X)／last(X)．VRO集合Xの最初／最後のVR実現値のdb鍵をδ_0とδ_Xに記憶する。

(2)　next(X)／prior(X)．レジスタδ内の値をdb鍵とするVR実現値を，δは指示するということにする。δ_Xの指示するVR実現値の次／前のVR実現値のdb鍵をδ_0とδ_Xに記憶する。

(3)　first(Z)／last(Z)．δ_Xの指示する親X内のVR実現値xのm(x)の最初／最後のVR実現値y∈Yのdb鍵をδ_0，δ_Y，δ_Zに記憶する。

(4)　next(Z)／prior(Z)．δ_Zの指示する子Y内のVR実現値yを含むm(x)のyの次／前のVR実現値のdb鍵をδ_0，δ_Y，δ_Zに記憶する。

(5)　owner(Z)．δ_Zの指示するVR実現値yを含むm(x)のxのVR実現値のdb鍵をδ_0，δ_X，δ_Zに記憶する。

(6)　get(X)．δ_Xの指示するVR実現値xのデータ項目値をβ_Xに記憶する。

(7)　any（X，v）．Xのあるデータ項目tが直接アクセス項目（CALC又はindex項目）のとき，t＝vなる最初のVR実現値のdb鍵をδ_0とδ_Xに記憶する。

(8)　dup(X)．(7)と同様であるが，δ_Xの指示するVR実現値xより大きい順

序のXのVR実現値yで，t(x)= t(y)のもので最小のyのdb鍵をδ_0とδ_Xに記憶する。

(9) accept（X，α）／accept（Z，α）．δ_X／δ_Zをレジスタαに記憶する。

(10) find（X，α）．αの内容をδ_0とδ_Xに記憶する。

(11) find（Z，α）．αの内容をδ_0，δ_Z，δ_Xに記憶する。□

以上の各演算が正常に終了したとき，レジスタrにS（uccess）が，そうでないときに，F（ailure）が記憶される。以上のVOPは，従来のDMLと一対一対応出来る。例えば，first(Z)は，COBOL DMLのfind first Y within Zである。更新演算VOPの意味の定式化は〔TAKIM 83 b〕に示してある。

5.2.5 巡航木反駁

本セクションにおいては，CNLによる問い合せをVNM上の操作演算VOPに変換する問題を考える。

5.2.5.1 目標グラフ

CNLのwffは，全てHorn節であるとする。節には，プログラム節と目標節がある。

〔定義〕 プログラム節は，次の形式

$$A \leftarrow B_1, \cdots, B_m \ (m \geqq 0) \cdots\cdots (2)$$

をしている。AとBi（$i = 1, \cdots, m$）は正リテラルである。正リテラルはR（$t_1 \cdots, t_n$）の形式で，Rはn項述語記号で，各t_iは定数又は変数である。Aを節の頭，$B_1, \cdots, B_m$を本体とする。$m = 0$のとき，事実節という。プログラム節の集合をプログラムとする。□

〔定義〕 目標節のGは，次の形式をしている。

$$\leftarrow B_1, \cdots, B_m \ (m \geqq 0) \cdots\cdots (3)$$

$m = 0$のとき空節という。□

目標節Gは問合せを表している。

〔定義〕 θを節集合P∪{G}の代入とする。 $(G_1 \wedge \cdots \wedge G_m)\theta$がPの論理的帰結（$P = ((G_1 \wedge \cdots \wedge G_m)\theta)$）であるとき，$\theta$をGの答代入という。□

θはG内の目標変数に対する代入で，問合せの目標集合の一つを与えている。

〔定義〕　プログラム内に，V記号Fを頭に含む節の集合

$$\{ F_i \leftarrow B_{i1}, \cdots, B_{il_i} \mid i = 1, \cdots, k, l_i \geqq 0 \} \cdots\cdots (3)$$

があるとき，Fを視野記号とし，(3)をその定義とする。各B_{ij}は，R，S，又は視野リテラルである。k＝1のときFを決定的，k＞1のとき非決定的であるという。□

目標節Gにおいて，接頭辞“？”が付されている変数を目標変数と呼ぶ。また，変数は大文字により，定数は小文字により表われているとする。

ここで，例えば図5.15において示したスキーマについて，節集合Fは次のように与えられる。

F＝{ 1) CE(X)<－E(X, c).
2) CE(X)<－DE(Y, X), E(X, T), D(Y, c).
3) E(a, c)<－.
4) E(b, c)<－.
5) E(c, t)<－.
6) E(d, r)<－.
7) D(1, c)<－.
8) D(2, a)<－.
9) DE(1, a)<－.
10) DE(1, c)<－.
11) DE(2, b)<－.
12) DE(2, d)<－.}

ここで，EおよびDはR記号，DEはS記号，CEはV記号である。節3～6は，従業員a，b，c，dが存在し，それぞれの専門がc，c，t，rであることを記述している。7～8は二つの課c，aがあることを示している。9～12は課cはaおよびcより，aはbおよびdから構成されていることを表現している。1～2は，技術者の専門がcであるか，または課cであるということを示している。

5.2.5.2　ＳＬＤ導出

有限節の導出法としてＳＬＤがあり，これを推論規則とした体系が健全であることが示されている〔APTR 82，LLOYD 84〕．目標節G（＝←G_1，…，G_m）に対して，ＳＬＤでは，ｉ＝１から順に，←G_i $\theta_1 \cdots \theta_{i-1}$ の答代入 θ_i を求め，$i=m$ で代入の合成 $\theta_1 \cdots \theta_m$ を目標変数の代入に制限した θ がGの答代入となる。θ_i が求まらないとき，←G_{i-1} $\theta_1 \cdots \theta_{i-2}$ の次の答代入 θ_{i-1} を求め（これをG_i からG_{i-1}への後戻りという）以上を繰り返す。

例として，目標節←A（x，y），B（y，ｔ），C(x)を考える。まず,←A（x，y）の答代入 θ_1＝｛x／ａ，y／ｂ｝が得られたとする。次に←B（y，ｔ）θ_1 =←B（b，ｔ）に対して θ_2＝｛ｔ／ｃ｝が得られ，←C(x)θ_1 θ_2 =←C(a)に対して，Ｐが事実節C(a)←を含まないときＢに後戻りする。しかし，この後戻りは，ＢとＣが変数を共有していないので無駄である。←B（b，ｔ）の全ての代入を得られてしまったとき，Ａに後戻りし新しい θ_1＝｛x／ｂ，y／ｃ｝を求め，Ｂで θ_2＝｛ｔ／ｈ｝が，C(b)に対してC(b)←とのmgu θ_3＝ϵ（同一代入）が得られ，答代入 θ＝｛x／ｂ，y／ｃ，ｔ／ｈ｝が求まる。この例で，ＣからＢへではなく，Ａに後戻りさせると効率がよくなる。この考えを進めたのが，次に示す巡航木導出である。

5.2.5.3　巡航木導出

次に，ＮＶＭの特徴を活かし，ＳＬＤ導出の改良を行う。巡航木導出の目的は，

(i)　不要なバックトラックの減少

(ii)　繰り返し，答代入を求めることの効率化である。

これらを実際に実現するための戦略について述べる。

目標節G（←G_1 …，G_m）内の部分目標変G_i が決定的，非再帰的視野のとき，定義F_i←G_{i1}，…，G_{ii} のF_i とのmgu（most general unifier）θを求め，←（G_1，…，G_{i-1}，G_{i1}，…，G_{ii}，G_{i+1}，…，G_m）θを目標節Ｇとする。Ｇ内に決定的非再帰的視野がなくなるまで，この操作を繰り返す。これは間合せ修正〔ＳTONM76〕と同じ操作である。

次に，このＧ（←G_1，…，G_m）に対して，以下の手続きGGでグラフ（目標グラフ）を構成する。

〔ＧＧ〕(1)　Ｇ内の各Ｒ／視野リテラルG_iに対して，各々Ｒ／Ｖ点G_iをつくる。G_iが目標変数xを含めば，G_iを目標点という。

(2)　Ｇ内に，リテラルＡ(x, …)，Ｃ(x, y)，Ｂ(y, …)が含まれ，ＡとＢはＲ述語記号，ＣはＳ述語記号のとき，ＡからＢに有向なＳ辺Ｃを設ける。

(3)　(2)以外でG_iとG_jが共通変数xを含めば，両点間に無向なＪ辺xを設ける。ＳとＪ辺を併せて結合辺という。□

次に巡航木を定義する。

〔定義〕　次の条件を満足する目標グラフを巡航木Ｔとする。

(1)　ある点を根とし，各点の子は左から右に順序付けられた順序木である。親と子の間の辺を主辺とし，他を従辺とする。

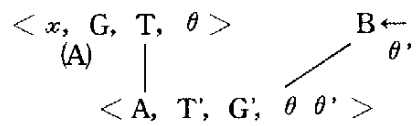
(2)　全ての従辺は，根と葉間の路内にある。

(3)　目標点は，木の最右の路内に全てある。根から最下の目標点までの路を幹とする。□

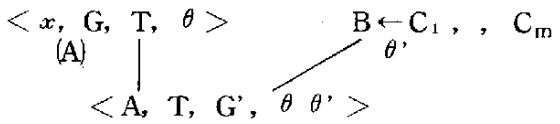
任意の目標グラフＧに対して，同一の答代入を得れる巡航木Ｔが存在することは，〔TAKIM 83 a〕に示されている。ＣからＴの生成方法を，次に述べる。さて，ＳＬＤでは，目標節内でのリテラルの全順序が導出順序を与えたが，巡航木では変数を共有したリテラルの間の導出順序を与えていて，リテラルを半順序付けている。

巡行木導出では，目標節は$<x$，Ｇ，Ｔ，$\theta>$で表現される。Ｔは基本式を点とした木（巡行木），xはＴの点，Ｇは目標グラフである。Ｒを計算規則とすると，Ｒ($<x$，Ｇ，Ｔ，$\theta>$)は，Ｇ内のある基本式Ａを選択する。

(1)　ＡがＳ又はＳ基本式のとき，CODASYL データベースをアクセスして，Aθと単一化可能な実現値をDML（e.g. find next … within …）を用いて検索する。ここで，G' はＧからＡを除いたグラフで，T' はＴにＡを加えた木である。



(2)　ＡがＶ基本式のとき。ここで，G' はＧからＡを除きC_1，　，C_mを加えたグラフである。



計算規則Rは，< x, G, T, θ >から，

(1)G内で，xと隣接する点のなかで，最小コストの点yを見つける。

(2)見つかれば，return(y)。　(3)見つからねば，xの親をxとして(1)へ。

xの親がなければ，G内の最小コストの点yを選択する。コストは，アクセスされるレコード実現値数の見積りから決定される。コスト計算については後述する。

このような導出において，< x, G, T, θ >で失敗した場合には，T内のxの親yでの目標< y, T', G', θ' >へ戻る。

Gが空となったとき，反駁に成功し，答え代入が求まる。繰り返し答え代入を求めるには，木Tを用いる。Tには，R，S，N基本式だけが含まれる。T内の各点xに対して，

n(x)=縦型順序によるxの次の点。

p(x)= xの親の点。

t(x)= xを親とする部分木の最後の点。

c(x)= xを親とする部分木内の目標点の数。

g(x)= xの左側で最近接の点。

このとき，xをTの最後の点とする。

(1)TH= x，TT= t(x)として，THからTTまでの点をTの縦型の順にCODASYLデータベースをアクセスしながら導出を行う。　(2) x = TTで成功すれば，TTの次の点の中で，c(y)> 0の点yを順に見つけ，x = yとして，(1)へ。　(3) xで失敗すれば，x = p(x)として，xに戻る。　(4) xがTHより前のとき，(1)へ。

5.2.5.4　コスト計算

巡航木の形成時に利用するコスト関数 fcost，ecost，cost を定める。コストの単位は，VNMが操作するVR実現値の数となる。

A.　fcost (B)

fcost (B)は，リテラルBの全答代入を求めるコストである。

(1)　BがVRO集合Xを指示するとき，χの基数 card(χ)とし，Xの項目

tの選択度〔HEVNA 78〕をσ_tとする。

(a) $fcost(B) = card(\chi)$ （順次アクセス）

(b) $fcost(B) = \sigma_t \cdot card(\chi)$ （直接アクセス）

(2) Bが視野で，導出が行なえるn個の視野定義$B_i \leftarrow B_{i1}, \cdots, B_{in}$（$i = 1, \cdots, n$）があるとき，

$fcost(B) = M \cdot \Sigma_{i=1, \cdots, n}\, cost(\leftarrow B_{i1}, \cdots, B_{in})$となる。

Bが非再帰的なとき，M＝1で，再帰的なとき，M＞1である定数である。Mの値を大きくすることにより再帰的な視野の導出を遅らせることが出来る。

B. ecost（S, B）

これは，Bの親B'の一つの答代入に対して，Bの全答代入を求めるコストである。まずVSO集合Zに対して，c_Zを親が持つ子の平均数（≥ 0），i_Zを子が親を持つ確率（≤ 1）とする。このとき，

$$ecost(S, B) = \begin{cases} c_Z & \text{if (A)} \\ i_Z & \text{if (B)} \\ fcost(B) & \text{otherwize} \end{cases}$$

ここで，(A)は，first(Z)およびnext(Z)によるアクセスの場合である。また，(B)は，owner(Z)によるアクセスを行う場合である。

C. cost(T)

cost(T)は，目標節Gの巡航木Tの最悪の場合のコストである。まず，図5.16に示す部分木Tを考える。Bは主辺Sで親と結合され，n個の部分木T_iを持ち，T_iの根はB_iでBと主辺S_iで結合されている（$i = 1, \cdots, n$）。このとき，tcost(T)は，根Bの一つの代入に対して，T内でアクセスされる全コストを与えるものとする。

$$tcost(T) = \begin{cases} 1 + \pi(B) \cdot [ecost(S_1, B_1) \cdot tcost(T_1) + \\ \quad \sigma(T_1) \cdot [ecost(S_2, B_2) \cdot tcost(T_2) + \cdots \\ \quad \sigma(T_{n-1}) \cdot [ecost(S_n, B_n) \cdot tcost(T_n)] \cdots]] & \text{if } n \geq 1 \\ 1 & \text{if } n = 0 \end{cases}$$

$$\sigma(T) = \begin{cases} \pi(B) \cdot 1 1_{i=1, \cdots n}\, \delta(S_i) \cdot \sigma(T_i) & \text{if } n \geq 1 \\ \pi(B) & \text{if } n = 0 \end{cases}$$

$$\delta(S_i) = \begin{cases} 1 & \text{if } ecost(S_i, B_i) \\ ecost(S_i, B_i) & \text{otherwise} \end{cases}$$

π (B) ＝ Bについての制限を，ＶＲＯ集合Ｘ内の各要素が満足する確率以上を用いると， cost (T)は，次式で与えられる。

$$\text{cost}(T) = \text{fcost}(B) \cdot \text{tcost}(T)$$

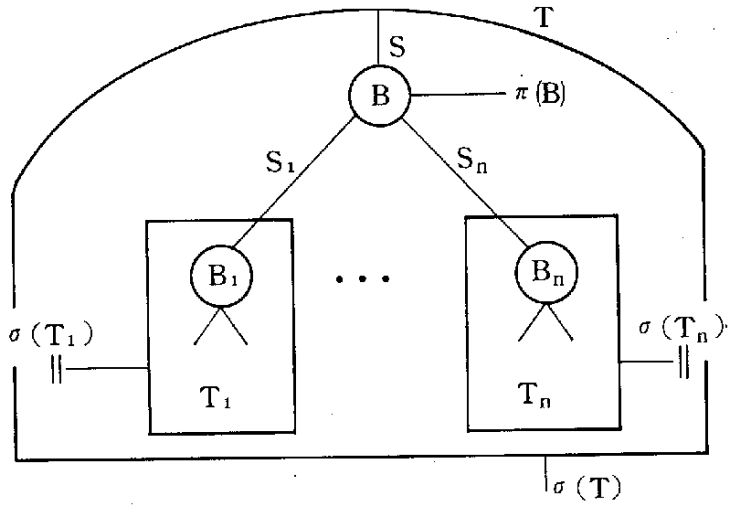


図 5.16　巡航木Ｔ

5.2.6　システム構成

以上の設計概念に基づいたシステムＬＩＰ（ＣＯＤＡＳＹＬ　ＤＢＳの論理インタフェースプロセッサ）の構成を，図5.17に示す。現在M－360ＲのＡＩＭ／ＤＢ，ＲＤＢ，ＵＴＩ－Ｌｉｓｐ〔ＣＨＩＫＴ81〕を用いて実現している。ＣＯＤＡＳＹＬ型のＡＩＭ／ＤＢを事実データベースとし，関係型のＡＩＭ／ＲＤＢには，視野定義と新事実節を格納するために用いている。ＶＯＰはＰＬ／Ｉプログラムとして実現し，Ｌｉｓｐの関数としている。

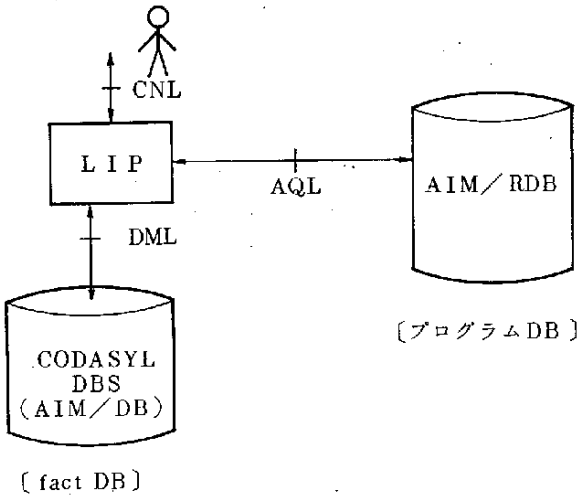


図 5.17　ＬＩＰの構成

5.2.7　システムの実現

現段階において，LIPはそのサブシステムとなる部分となるVOPおよびDD／Dを実現している。VOPは，PL／Iサブルーチン群として定義されており，このサブルーチン群を実行可能とするLisp関数群を用意している。また，LIPシステムでは，システム内部にDD／Dを用意することによりコスト計算およびタイプのチェック等を行うことを可能とする構成となっている。

本セクションにおいては，それぞれの部分について述べる。

5.2.7.1　VOPの実現

VOP，AIM（富士通提供CODASYL型データ・ベース）およびLispの関係を図5.18に示す。

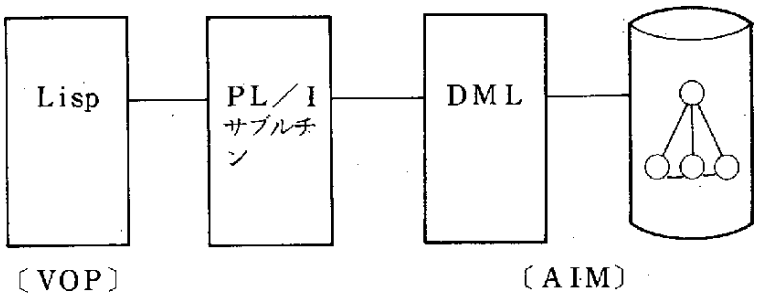


図5.18　システム構成

まず，VOPを実現するために次のような目標とそれに対する制約条件について考えることにする。

〔目標〕

(i)　Lisp関数として利用可能

(ii)　高効率性

(iii)　スキーマ独立

〔制約条件〕

(i)　AIMは，PL／I，COBOL，アセンブラをホスト言語としている。

(ii)　UTI－Lispは，FORTRANおよびアセンブラのインタフェースを有している。

(iii)　VOPにおける関数群は，CODASYLデータ・ベースを独立して操作するのではなく，関数適応以前の状態を保持する必要がある。

このような目標および制約条件のもとで，ＶＯＰを構築することにする。ここで，ＬＩＰシステムを実現するためにUTI－Lispを用いることの理由は，ＵＴＩ－Lispが富士通提供Lisp 1.5 に比べて実行速度が格段に速いこと，Lispがサポートするデータ構造が非常に柔軟であるのでグラフ構造等が容易に記述できること，等の理由による。

まず，我々は言語間インタフェースについて検討した。Lisp－DML結合では，Lisp－ＰＬ／Ｉ結合がＬＩＰの場合には適当であると判断した。理由としては，Lisp－ＤＭＬ間結合のために中間言語を種々設けることは，その動作環境設定の効率の悪さから，非常に実行速度が低下することが確認されたことである。例えば，Lisp－Fortran－ＰＬ／Ｉと結合し，富士通提供による言語間インタフェース機能を用いると，約１回のＰＬ／Ｉ呼び出しに0.6秒要する。この数値は，FACOM　Ｍ360Ｒを用いて実験したものである。したがって，われわれは，Lisp－ＰＬ／Ｉを直結することを考えてみた。その結合方法としては，DML（Data Manipulation Language）のホスト言語の一つであるＰＬ／Ｉロードモジュールを ＦＯＲＴＲＡＮとみせて（ＰＬ／Ｉのオプション機能），Lispライブラリとして定義し，Lisp関数の一つとしてVOP演算群を呼び出し可能な状態にする。ここでは，ＰＬ／Ｉソースプログラムは，サブルーチン形式になっている。これの形式は後に述べる。

次に問題となったことは，スキーマの独立性および現在子保存の問題である。前者のスキーマ独立性の問題であるが，現段階においては解決されていない。なぜならば，ＰＬ／Ｉは動的にデータ構造を変えることが不可能であり，スキーマはあらかじめ固定せざるを得ないからである。次に，現在子保存の問題であるが，これはLisp側より見れば，ＰＬ／ＩのVOPモジュール群プログラム（VOPM）に静的な記憶域を用意し，VOPMが一旦呼びだされれば，容易にパージ（切り離す）されないことで解決している。個々のVOP演算は，このVOPMのサブルーチンとして定義されている。Lisp側ではこのVOPMに各機能のパラメタを付けて，呼び出す形式となっている。VOPMを呼び出すためには，引き数は３個である。第１引数はVOP演算名，第２引数はＶＯＰ演算実行のための引数，第３引数は値の受け渡しのための引数である。したがって，Lisp側における第３引数はＶＯＰ関数実行結果のための引数である。図5.19にその第３引数のため

のデータ構造を示す。ここで，タイプ（E，S，I，N，C）は，それぞれアイテムが定義されていないというような簡単なレベルのエラー，命令が実行できないというような重大なエラー，整数，浮動小数点表現による実数および文字であることを示している。実際的な値は，第2カラム目より左づめにより受け取る。表5－2に，VOP関数実現のためのPL／Iサブルーチンおよび Lisp 側における定義を列挙する。なおこれらの Lisp 関数の関数値は，先に述べた第3パラメタを，それぞれのタイプに従い整理されたものになっている。ただし，エラーの場合には，値は Nil にまとめられている。また，VOPに対する引数は，前述したVNM上のVOPと同一なものである。

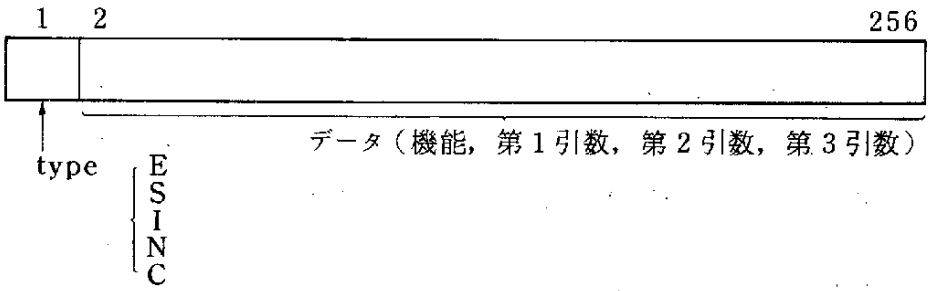


図 5.19　受け渡し構造

表 5.2　VOP関数

VOP	LISP	PL／I
注1	V#DBINIT	DBINIT
注1	V#DBTERM	DBTERM
first (X)	V#RECFST	RECFST
next (X)	V#RECNXT	RECNST
next (Z)	V#SETNXT	SETNXT
prior (Z)	V#SETPRP	SETPRP
accept (X, α)	V#RECACT	RECACT
accept (Z, α)	V#SETACT	SETACT
find (X)	V#RECFND	RECFND
find (Z)	V#SETFND	SETFND

VOP	LISP	PL／I
owner (Z)	V#OWNER	OWNER
any (X, v)	V#ANY	ANY
dup (X)	V#DUP	DUP
get (X)	V#GET	GET
注2	V#GETITM	GETITM

注1：VOPには定義されていないが，実際にデータベース(AIM)のオープン／クローズ

注2：実際にはレコードの項目単位の受け取りが必要である。

5.2.7.2　DD／Dの実現

DD／Dは，LIPシステム内におけるデータとして取り扱われている。通常の場合，DBMSにおける機能として有しているが，本システムでは図5.20に示すような関係になっている。LIPにおいて取り扱うDD／D情報は，システム起動時に，ファイルより構築される。むろん，LIPシステムの機能の一部としてDD／Dエディタを用いることにより，DD／D情報の定義更新を容易に行うことができる。

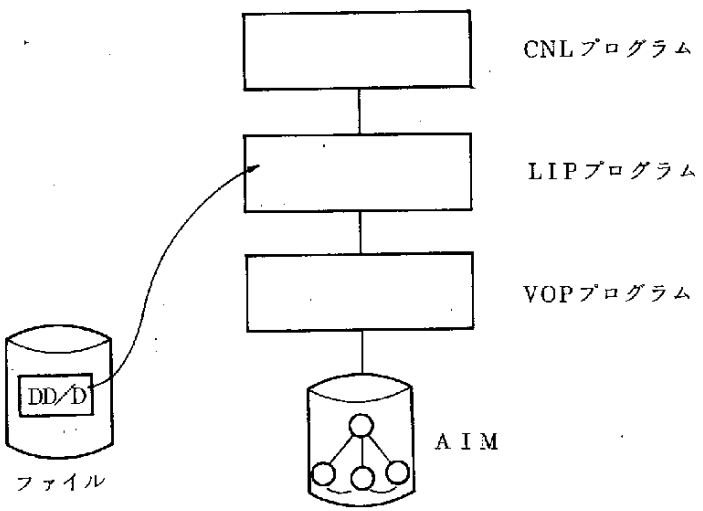


図5.20　DD／Dの位置付け

```
> (DB-PRCD-RECORDTYPE)
 RECORD-TYPE: EMP
 ITEM-NAME    FORMAT    NULL    CARDINALITY    SELECTIVITY
 ADRESS       (C 100)   N       NIL            NIL
 SAL          I         N       NIL            NIL
 AGE          I         N       NIL            NIL
 ENAME        (C 20)    NA      NIL            NIL
 ENO          I         NA      100            1
 AREA-NAME: DB-AREA   CARDINALITY:100   LOCATION-MODE: (S DE)
```

(i) レコード型

```
 SET-NAME:                  DE
 OWNER:                     DEPT
 MEMBER:                    EMP
 MEM-TYPE:                  OM
 CONNECTIVITY:              +0.45000E+01
 I-CONNECTIVITY:            +0.80000E+00
 SORT-TYPE . SORTITEM:      A . ENO
 DNA-MODE:                  (ENO)
```

(ii) セット型

図 5.21　DD／Dエディタの出力例

DD／D情報をＡＩＭに存在するDD／Dを用いなかった理由は，次のことである。

(i) VOP実現と同様にスキーマ独立が困難であったこと。

(ii) 将来的に，ＤＤ／Ｄに知的な機能を付加することができる可能性が生じること。

現段階における，ＤＤ／Ｄに格納されている情報，およびそのデータ構造を（Ｓ式）記す。図 5.20は，DD／Dエディタを用いて出力された，レコード型およびセット型実現値に関するDD／D情報である。

図 5.21において示されている内容より，各レコードにおける項の実現値の型，コストを求めるためのセット型における平均の子の数，親の数等を求めることができる。

5.3 まとめ

本セクションにおいては，主に次のことについて述べた。

(i) エキスパート・システムとそのアーキテクチャ。データ・ベース・システムとの融合の必要性。

(ii) 知識表現モデルであるフレーム・モデルと階層型データ構造の定式化。フレーム・モデルを階層型データ構造とみることにより，第一階述語論理により記述でき得ることを示した。

(iii) 従来のCODASYL型データ・ベースに対する高水準インタフェースＬＩＰの概略。LIPを構成するために，従来のCODASYLモデルを再構成した。

一般に，知識ベース技術の応用としてのＣＡＤシステム，大規模医療診断システム等においては，データ・ベースを利用した技術は必要なものとなってくるであろう。現段階において，知識型システムは，比較的小さな問題分野に適応しかつその分野においては大量な知識を用いるが全体としての量は少ないという場合が少なくなかった。しかしながら，統合化CAD〔OHSUS 85 b〕のような場合を考えた場合，データ・ベース技術はエンジニアリング・データ・ベースと呼ばれている分野と相まって考えなければならない問題が多々あると思われる。例えば，知識ベースの管理とデータ・ベースの管理，これをどのようなメカニズムにより整合性のある情報管理メカニズムを構築するのかということである。互いの知識が相矛盾するような場合に，どのような推論メカニズムを適応するのかということが問題となるであろう。また，事実に関する情報がデータ・ベースに格納されている場合，当然情報の検索空間は多大なものとなるであろう。これをどのように制御するのか，さらにその制御の効率的な面で実用システムとして耐え得るのかという問題も生じる。多大な外部システムへのアクセスが予想されるので，効率は悪くなるであろう。また，データ・ベースを一部に有することによりより高度な知識情報処理システムの開発が期待されるであろう。例えば，データ・ベースからの知識獲得のような機能である。

次の問題として，知識型システムの評価という観点において，客観性のある定義は意味のあることであると考えられる。なぜなら，そのシステムの性質が明らかとなるからである。特に，汎用ツールの場合，システムの性質が明らかになるということは，応用分野への適応ということとはまた異なった意味合いを有する評価が可能となる。例えば，どのように検索された結果ならば論理的な意味において保証されているか，またその汎用ツールがどのような体系にそ

っているのかを明らかにすることができる。このような意味において，汎用フレーム型データ構造をサポートするZEROの定式化は，必要なことであった。今後の課題，拡張方向として，次のようなことが挙げられる。

(i) Part－of リンクの表現の意味

AKOリンクおよびPart－of リンクの意味の表現は，知識表現モデルにとっては重要な問題である。特に物理的な対象（例えば，自動車，人間，プログラム）の構造記述のためには，システムがPart－of リンクをサポートすることが可能でなければならないと思われる。ＣＡＤ，プログラム合成においては，この構造記述が中心になることが言われている〔UENOH 85〕。

(ii) フレーム・システム（フレームデータ構造）に対する更新，副作用の取り扱い。

第１階述語論理では，ドメインの更新または削除というようなことは考えられない。しかしながら，フレーム・データ構造を用いた知識型システムでは，この構造を単なる情報格納用データ構造としてではなくフレームの内部状態（例えば，スロット値の更新，付加，削除）を変化させながら推論を進めると考えるほうが適切である。これは，第１階述語論理においては，ドメインの変化ということに対応している。

ZEROにおけるフレーム・モデルに対する副作用としては，例えば，階層の上位フレームにおいてあるスロットの値を削除するとする，この時階層の下位に位置するフレームに相当するスロット値も削除されること，または，クラス・フレームの削除はそのフレームをノードとする部分木も削除されるということ，である。このような動作をいかに記述するかということは，１つの問題点となっている。これの解決が，当面の課題の一つである。

データ・ベースと知識ベースの結合という意味において，ＬＩＰは１つの試みである。従来のCODASYL DBSに対して論理型言語CNLをＬＩＰ，は提供している。CNLプログラムは（制限された）Prologプログラムステートメントである。例えば，CNLにおいては現段階においてはCODASYL DBに対する更新が考えられていないため，Prologにおけるassert，retract等の（Prologにおける）演算は提供されていない。この問題については，後に述べる。

現在ＣＮＬプログラムをインタラクティブに評価を行っている。しかしながら，再帰的な定義がない場合には，ＤＭＬプログラムにコンパイルするこ

とができる。再帰的な定義がある場合には，DMLは現在子に基づいているので，現在子の保存のためのスタックが必要であろう。ここでは，DMLプログラムがCODASYLデータ・ベースの巡航的アクセスを基本としていることから，演　に必要な論理式評価のためのアクセスを行うインタラクティブな規則が有効であると考えている。

従来，論理データ・ベースに係わる問題（例えば，再帰的視野，定理証明システムと結合した一貫性，無矛盾性の検討，問い合せ評価，知的なバックトラッキング等）は，関係型データ・ベースを用い行なわれてきた，CODASYL DBSも導出に適合していることを示した。この時，CODASYL DBSの特徴である巡航によるアクセスを有効に利用する巡航木導出なる導出を適合することにより，単純な深さ優先検索，およびバックトラックキングの制御を行うことができる。巡航木導出は，この種の制御を行っており，検索の最適化および効率のよりバックトラックキングを達成している。

ＬＩＰに対する今後の課題としては，次のことが挙げられる。

(i) 理論的背景および実現方法が整備検討された。次には，その実働化である。

(ii) 知的バックトラックキング〔CAMPJ 84〕およびCNLプログラムコンパイル方式の検討。

(iii) CODASYL DBSに対する更新演算の定義，理論的背景等の整備，および巡行木導出の改良。先に述べたように，現段階においては，新しい事実は，AIM／RDBに格納される構成になっている。この新事実が付加された場合に，どのように対応するのかということについては，今後の課題である。

最後に，本セクションでは今後課題となっていくであろう知識ベースとデータ・ベースの結合という点について述べた。今後，これの発展形として，知識ベースとデータ・ベースの融合に関するアプリケーションを通じての評価も必要であろうと考えている。

6. まとめと今後の課題

本報告書は，本事業の最終年度として，事業全体の成果をとりまとめたものである。

本事業では，以下の作業を行った。

(1) ローカル情報システム（ＬＩＳ）の設計，開発

a. 基本通信手段としてのトランスポート群制御プロトコル（TCCP）

b. DDBSを実現するための分散型データ・ベースプロトコル（DDBP）

(2) パーソナルコンピュータ用関係型DBMSの設計，開発，改良

a. PRDBMS

(3) 高水準インタフェースーＬＩＰの設計，開発

(4) 異種のDBS間でのインタオペラビリティをはかるための統合化技術の検討

第１年度では，(1)のＬＩＳの全体モデルの確立及び概念設計を行ない，又(2)のパソコン用関係型DBMS PRDBMSの設計及びPC 9800 上にプロトタイプを実現した。第２年度は，(1)－ａのTCCPの設計及び(1)－ｂのDDBPの設計を行ない，ＬＩＳの基本通信としての機能を実現した。またPRDBMSの改良を行ない，PC 9800に加えて，ワークステーションSUN 2／120 上で実働化した。さらに，(4)のＬＩＳの同種化，統合化技術を検討し，代数によるDBSの同種化，安全性の制御を提案し，放送通信を用いた，効率のよい分散トランザクション処理について設計を行なった。最終年度では，(1)－ｂのＤＤＢＰの実現を行ない，ＬＩＳのDDBSとしての機能を確立した。パーソナルコンピュータ用関係ＤＢＭＳでは，PRDBMSの改良版であるPRDBMS－Ⅱの設計及び開発を行ない，SUN 2／120上で実現した。さらに最終年度では，(3)の論理型言語インタフェースＬＩＰの設計，実現を行なった。ＬＩＰはCODASYL DBS（AIM）上の論理的インタフェースであり，この実現により，利用者は従来より簡単で，強力な検索が可能となっている。

本事業では，以上述べたように，また１章で述べたように，分散型情報システム，さらに一歩進んで，統合化知識情報システム（分散した情報をより知的かつ一元的に扱えるシステム）を目指し，その基礎となる部分を提案，設計，実現したものである。

現在，ＬＩＳは分散型データ・ベース・システムとして，SUN 2／120 のDBS PRDBMS Ⅱを含めた形で，その機能を果している。又，パーソナルコンピ

ュータ用関係DBMSはPC 9800とSUN 2／120 上で実働している。さらに高水準インタフェースＬＩＰは汎用計算機（Ｍ－360Ｒ　FACOM）上のCODASYL型のDBMS，ＡＩＭ（ FACOM ）のインタフェースとして，実現されている。

今後の課題としては，

(1) PRDBMSⅡのPC 9800上への移殖，改良

(2) ＬＩＳのDDBP，TCCPの改良，高速化

(3) ＬＩＰの関係型DBMS（例えばＡＩＭ／RDB，PRDBMSⅡ上での実現

(4) 既存のデータ・ベースの知識ベース化の方法の検討

などがあげられる。

将来のより知的な分散型知識情報システムの実現を目指して，その基盤となる技術の検討，プロトタイプ等の開発をこれからも行なっていくつもりである。

〔ABITS 84〕 Abiteboul, S., Hull, R., IFO:A Formal Semantic Database Model (Preliminary Report), Proc. ACM SIGACT-SIGMOD. Symposium on Principles of Database Systems, 1984

〔APTR 82〕 Apt, R., and van Emden, M. H., "Contributions to the Theory of Logic programming," JACM, Vol. 29, No. 3, pp. 841-862, 1982

〔BRACR 79〕 Brachman, R, J., On the Epistemological status of Semantic Networks, Associative Networks, (Findler, N. V. ed), Academic Press, 1979

〔BRACR 85〕 Brachman, R. J., Schmalze, J. G., An Overview of the KL-ONE Knowledge Representation System, Cognitive Science 9, 1985

〔CACM 85〕 Communications of the ACM, No. 9, Vol 25, 1985

〔CAMPJ 84〕 Campbell, J. A., "Implementations of Prolog," Ellis Horwood Limited, pp. 175-278, 1984.

〔CHANC 81〕 Chang, C. L., "On Evaluation of Queries Containing Derivec Relations in a Relational Data Base," Logic & Database, Plenum Press, 1981.

〔CHIKT 81〕 Chikayama, T., UTILISP Manual, University of Tokyo, 1981

〔CLOCW 84〕 Clocksin, W. F. and Mellish, C. S., "Programming in Prolog(2nd ed)", Springer-Verlag, 1984.

〔CODAS 73〕 CODASYL, "CODASYL Data Description Language Journal of Development," June 1973 〔NBS Handbook 113 (1974)〕.

〔CODAS 78〕 CODASYL DDL Committee, "Report of the CODASYL Data Description Language Committee," Information Systems, Vol. 3, No. 4, 1978, pp. 247-320.

〔CODDE 70〕 Codd, E. F., "A Relational Model of Data for Large Shared Data Bank," Communications of the ACM, Vol. 13, No, 6. June 1970, pp. 337-387.

〔DATEC 81〕 Date, C. J., An Introduction to Database Systems, 3 ed, Addison-WeeLey, 1981.
〔DAVIR 83〕 Davis, R., Shrobe, H., Representing Structure and Behavior of Digital Hardware, IEEE Computer, No. 10, 1983.
〔DAYAU 82〕 Dayal, U., et al., "Query Optimization for CODASYL Database Systems," Proc. of the ACM SIGMOD, 1982, pp. 138-150.
〔ENDEH 72〕 Enderton, H. Mathematical Introduction to Logic, Academic Press, 1972.
〔FEIGE 77〕 Feigenbaum, E. A The Art of Artifieial Intelligence : Themes and Case Studies of Knowledge Engineering. IJCAI 5, 1101-1029, 1977.
〔GALLH 84〕 GALLAIRE, H MINKER, J. NICOLAS, J "Logic and Databasses : A Deductive Approach" ACM Computing Surveys, Vol. 16, No. 2, June 1984 pp. 153-185.
〔HELDG 76〕 Held, G. D., et al., "INGRES-A Relational Data Base System," AFIPS Conf. Proc., 1976, pp. 409-416.
〔HEVNA 78〕 Hevner, A. R. and Yao, S. B., "Query Processing on a Distributed Database," Proc. 3 rd Berkeley Workshop on Distributed Data Management and Computer Networks, San Francisco, California, Aug. 1978, pp. 91-107.
〔IEEE 83〕 IEEE Computer, Special Issue on Knowledge Representation, No. 10, 1983.
〔ITOH 86〕 Ito, H., Ueno, H., ZERO : Framet Prolog, to appear in LPC' 85, LNCS, Springer-Verlag
〔JACOB 82〕 Jacob, B. E., "On Database Logic," JACM, Vol. 29, No. 2, 1982, pp. 310-332.
〔JYOHO 85〕 情報処理, No. 12, Vol. 26, 1985.
〔KELLC 86〕 Kellogg, C., From Data Management to Knowledge Management, IEEE Comuter, Jan., 1986.

〔KOBAI 85〕 Kobayashi, I., "Classification and Transformations of Binary Relationship Schemata," Sunno Institute of Business Administration, 1985.
〔KOWAR 79〕 Kowalski, R., "Logic for Problem Solving," North-Holland, 1979.
〔LID 84〕 Li, D., "A Prolog Database System," Research Studies Press, 1984.
〔LLOYD 84〕 Lloyd, D., "Foundation of Logic Programming," Springer-Verlag, 1984.
〔MCDED 80〕 McDermott, D., Doyle, J. Non-Monotonic Logic I, Artificial Intelligence, 13, 1980.
〔MINSM 75〕 Minsky, M., A Framework for Representing Knowledge, The Psychology of Computer vision (Winston, P. H., ed), McGrow-Hill, 1975.
〔OHSUS 85a〕 Ohsuga, S., Yamauch, H., Multi-Layer Logic-A Predicate Logic Including Data Structure as Knowledge Representation Language, New Generation Computing, 3, 1985.
〔OHSUS 85b〕 Ohsuga, S., Conceptual Design of CAD System Involving Knowledge Bases, Knowledge Engineering in Computer-Aided Design (J. S. Gero, ed), North-Holland, 1985.
〔OHSUS 83〕 Ohsuga, S., "Developing a Deductive Relational Database for Uniform Handling of Complex Queries," JIP of IPSJ, 1983; pp. 123-137.
〔OLLET 78〕 Olle, T., "The CODASYL Approach to Data Base Management," John Wiley & Sons, 1978.
〔RITER 78〕 Reiter, R., On Closed World Data Bases, Logic and Data Bases (Gallaire, H., Minker, J., ed), Plenum Press, 1978.
〔RITER 80〕 A Logic for Default Reasoning, Artificial Intelligence, 7(2), 1980.
〔ROBIT 65〕 Robinson, J. A., A Machine-Oriented Logic Based on the Resolution Principle, JACM, 12, 1, 1965.

〔SHORE 76〕 Shortiffe, E. H. Computer−based Medical Consultations : MYCIN, North−Holland, 1976.
〔SMITD 85〕 Smith, D. E. and Genesereth, M. R., "Ordering Conjunctive Queries," Artificial Intelligence, Vol. 26, 1985, pp. 171−215.
〔STEFM 79〕 Stefik, M., An Examination of a Frame Structured Representation System, 6 th IJCAI, 1979.
〔STONM 76〕 Stonebraker, M., Wong, E., Kreps, P., and Held, G., "The Design and Implementation of INGRES," ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976, pp. 189−222.
〔SZOLP 78〕 Szolovits, P., Pauher, S. G., Categorical and Probabilistic Reasoning in Medical Diagnosis, Artifieial Intelligence 11, 1978.
〔TAKIM 80〕 Takizawa, M., et al., "Query Translation in Distributed Databases," Proc. IFIP '80, Tokyo−Melbourne, Oct. 1980, pp. 451−456.
〔TAKIM 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 AL 81−22, 1981.
〔TAKIM 82〕 Takizawa, 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−V15）の設計と実現" IPSJ Vol, 23, No. 3, 1982, pp. 665−675.
〔TAKIM 83a〕 Takizawa, M. and Noguchi, S., "CODASYLデータベースに対する非手続的更新インターフェースの基本概念"情報処理学会論文誌, Vol. 24, No. 6, 1983.
〔TAKIM 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.
〔TAKIM 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.

〔TAKIM 84〕 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, 1984.
〔TAKIM 85〕 滝沢誠, 盛屋邦彦, 伊藤秀昭, "CODASYL, データベースの論理型インタフェースLIPについて", アドバンスドデータベースシンポジウム, 1985.
〔UENOH 85〕 上野晴樹：知識工学入門, オーム社, 1985.

昭和６１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　タケミ印刷
東京都千代田区神田司町２－16
TEL （254）5840