情報システム 21－1
（1988. 11. 15）

データ中心システム設計
――その必然性と可能性――

堀内 一
（株）日立製作所コンピュータ事業部
東京都品川区南大井6－27－18

データ中心アプローチ（DOA: Data Oriented Approach）とは、先に共有資源としてのデータを設計し、そのデータに基づきながらソフトウェアを設計することを指す。共有資源としてデータを標準化し、そのデータ側から決定できるプロセスは可能な限りデータ対応に実現することを狙う。このようなソフトウェア実現方法はカプセル化（Encapsulation）とも呼ばれる。 カプセル化により、単純で明快なシステム要素を導くことができることから、システムの制御方式の単純化にも大きな効果をもたらす。
本稿では、DOAの背景、必然性を述べると共に、カプセル化の方式とカプセルによるシステム制御方式についても述べ、DOAによるソフトウェア開発のあり方についても述べてみたい。

Data Oriented System Design Methodology
An approach for constacting information system
using encapsulation technique

Hajime HORIUCHI
Computer group, Hitachi Ltd.
6-27-18 Minamiohi Shinagawa-ku, Tokyo 140, Japan

The techniques for data standardization featuring nomalization theories are recognized as tools for the elimination of data redundancy in data bases. The standardization of data has much more significances on information system design, since it provide the bases for standardizing programs. The specific thechnique which intends to encapsulate data and process into a particular package can be effective on designning clear software components. In this article, the concept and necessity of data oriented approach, are described and the ideal system structure which uses encapsulation techniques are also illustlated.

1　データ中心アプローチの背景と必然性

1．1　はじめに

情報またはデータを企業経営資源の一つと見なす概念は、７０年代末からＩＲＭ（情報資源管理）またはＤＲＭ（データ資源管理）として知られている。その概念は、単にデータや情報の有効活用だけでなく、既存システムの再構築、ＯＡ機器とメインフレームとの統合、あるいは戦略的情報システム（ＳＩＳ）の構築などを果す上でも有効なものと考えられている。　しかしながら、これまでのソフトウェアシステムの設計・開発に関する方法論は共有資源の存在を認めたものとは言い難く、ソフトウェアそのものの重複を安易に許してきた。その結果、ソフトウェア相互の不整合や不必要な依存関係を生じせしめ、不要なメンテナンスを余儀なくさせてきたと言える。

データ中心アプローチ（以下ＤＯＡと呼ぶ）は、システムやソフトウェアの開発に当たりデータを先に共有物として標準化し、それを基にソフトウェアの構造を導く方法の総称である。そこにはデータが共有資源であるという認識が存在し、共有資源の一貫性や完全性を重視する価値概念が優先する。

もとよりシステム開発は、要求されるシステムを求められる品質で、安く、早く実現することによって評価されるものと言える。しかし、単にソフトウェアの見掛け上の生産性向上を目的とする方法論では、システム資源の検討と手配が開発プロジェクトごとに個別に行われ、結果として資源重複を避けられない。その重複は、データとソフトウェアの相互関係の複雑化をもたらし、相互に変更を波及させる要因を増加させるものとなる。

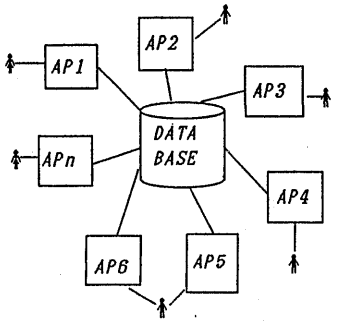
1．2　情報システム構築方法論への要請

７０年代にソフトウェアエンジニアリングの名の下に普及した構造化手法の多くは、システム化すべき対象業務を機能集合としてとらえ、その機能の分解によってモジュールを導く方法をとってきた。その中で、７０年代後半にデータ中心型の設計技法として登場したジャクソン法やオア法は、ソフトウェア構造を入出力のデータ構造から導く方法を示した。しかし、どちらも一つの企業や組織におけるデータ資源の全てを網羅的に捕らえてモデル化し、標準化する発想は持っていなかった。共有資源としてのデータを持つシステムの設計方法論としては不十分といわざるを得ない。

データを先に共有資源として標準化する発想を持つ技法は、８０年代に入ってからインフォメーションエンジニアリングの名の下に登場した。　データ中心発想の必要性が真に認識される様になったのが各企業で基幹業務のコンピュータ化を一巡した後だからである。コンピュータ化の領域に関する量的な拡大が、システムそのものに質的な変化を伴ったからである。

図１に示すものは、これまでの方法論によってもたらされる典型的なシステム構成である。一つの共有資源の周りに多数のプログラムが作成されている。時には、一つのデータベースの周りに数百本のプログラムが作成されることもある。要求が発生する都度、個別にプログラムが設計され開発されるからである。そのようなシステムで、一つのプログラムに変更が加えられると、予期せぬ多数のプログラムの変更を引き起こし、その対策に振り回されることになる。今日、システムの保守コストを引き上げているのは、ソフトウェアが見えにくいとか、読みにくいということよりも、このような劣化したシステム構造による不必要な変更余波（ripple effects)に起因することが多い。

- ***更新処理の分散***
- ***更新制約一貫性の維持が困難***
- ***変更の波及（余波効果）***



図－１　伝統的手法によるシステム構成

システム開発の方法論も、その時々のシステムの特性や環境条件に大きく依存する。即ち、コンピュータ化すべき領域が大きく残されていて、その充足が最優先とされる段階の方法論と、既に、膨大なソフトウェア資産を保有している段階の方法論は異なるのが当然である。

今日、多くの企業は、既に１０年、２０年という期間に亘るコンピュータ利用経験をもっており、そのソフトウェア資産の量も膨大 なものとなっている。

したがて、共有資源を持ちながら多数のプログラムから構成される企業情報システムを構築する方法論と、共有資源を持たずにパーソナルな領域の問題解決目的とするパソコンやワークステーションによるソフトウェア作りの方法論とは大きく異なるべきである。

つまり、共有資源をもつ企業情報システムはシステム開発における構成制約として、単に外部要求の充足や外見的明瞭性だけでなく、システムの内部的構造からの制約を意識すべきである。即ち、システム構造を決定するものとなる、プログラムとデータの相互関連、プログラム相互の関連などを複雑化させることなくシステムを構成できるものでなければならない。また、要求の増加や多様化に従って、ソフトウェア規模の増大や多様化を来さないようなパラダイムも確保されなければならない。

それは丁度、製造業が小規模な段階では、工程管理を中心とした生産形態から開始し、やがて製品点数が増大してくると部品中心の多品種量産形態へと進化するのと似ている。管理の主眼は工程（プロセス）から部品に移行する。多様化する最終製品対応に製造ラインを用意することよりも、共通化された部品対応に製造ラインを用意する方が効率的だからである。

情報システムの構築も、小規模の内はプロジェクトによる受注生産形態でソフトウェア生産を行うことで対処できるが、その拡大に伴いユーザも情報要求も多様化すると、最終製品としての情報を効率的に生産できる多品種量産の形態が要請されてくる。ソフトウェア生産よりも情報生産のための形態が望まれ、管理の主眼もプロセスから情報生産のための部品としてのデータに移行する。

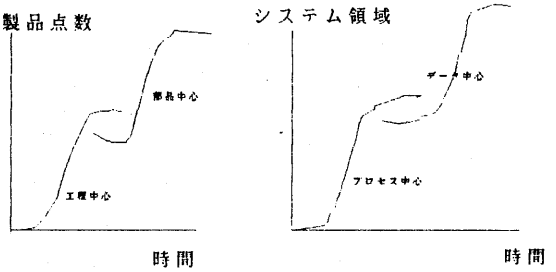


図　２　　生産形態の進化

基幹業務システムの構築を一巡し、既に膨大なソフトウェアを保有する企業が、システム方法論として求めているものは、次のような課題を解決してくれるものであろう。

（１）戦略的システムの構築に耐えられるような柔軟な情報システムの構築
（２）ソフトウェア規模の拡大に伴うシステム複雑性増加の抑止
（３）拡大するシステム利用者及び分散化するシステムに関する統制手段としてのインフラストラクチャ（基盤）の確立
（４）情報生産に関する多品種量産の確立
（５）ソフトウェア生産の工業化（ＣＡＳＥ基盤の確立）

ＤＯＡは、これらの要請に応えるべく、方法論の具体的目標を次のような課題に置いている。

（１）共有資源としてのデータの標準化
（２）標準データに基づくプロセスのカプセル化
（３）カプセル化による制御基準の明快化
（４）カプセルに基づく一貫性独立の実現
（５）データの部品化による情報生産の効率化

１．３　システム再構築とＤＯＡ

今日、情報システム構築の関心の多くは競争優位（competitive advantage）を確保するための戦略的情報システムに集まっている。しかし、戦略的な発想を持ちながらも容易にその実現を果たせない企業も多い。その理由の一つは、これまでに築いたシステムに一切の変更を加えられない硬直性にある。そ

の原因は、劣化した構造に起因する低い保守性にある。システム変更の余波効果を最小限にとどめるために、劣化した既存システムを捨て、新たに再構築する動きが強いのもそのためである。　金融機関などに見られる第３次オンラインはそのような目的を含んだ再構築と見ることができる。一金融機関当り七百億から１千億円の費用を投資しての再開発は、とりも直さず、激しい金融戦争を勝ち抜くための柔軟性をシステムに保持させながら、競争優位性を確保するために他ならない。

再構築は、既に構築されている情報システムを、単にもう一度作り直すことを意味するものではない。システム全体に関する新しい体系をフレームワークとして定義すると共に、システム毎に保有されているデータ資源を抽出して、共有資源として適切に管理しその重複を排除できなければならない。

ＤＯＡは、まず共有資源としての適格性をデータに具備させることを目的にその標準化に重点を置く。また、その過程で一つの企業におけるデータの体系をデータモデルとして捉え、情報システムに対する安定的なフレームワークを構築する。ソフトウェアの多くはこのフレームワークに沿って、つまりデータ体系を遵守しながら再構築される。

１．４　システム複雑性増加抑止とＤＯＡ

既に述べたように、システムの複雑性を表す尺度の一つは、プログラムとデータの相互関連であろう。データとプログラムの設定が自由放任の状況にあれば、それらの相互関係は複雑化するだけである。

データを適切な管理の下に置き、その標準化を徹底し、重複を排除したとしても、プログラムの設定が適切に統制されない限り複雑性を下げることはできない。

製造業で、たとえ部品の標準化と共通化を徹底したとしてもと、それに合わせて工程の見直しと再編成が行われない限り、生産活動そのものの重複を排除することはできない。同様に、データを資源として標準化しても、それに合わせてソフトウェアが見直されプログラム構成までを標準データに合わせない限り、システム構造の簡素化を図ることはできない。

データベース設計の究極の指導理念は「一つの事実は１カ所にしか記録しない（ one fact single place)」にある。同様にソフトウェアも「一つの機能は１カ所にしか実現しない（ one function single place)」をその設計理念とすべきである。

但し、ここで機能と呼ぶものは、単にソースコードとしての類似性で判断されるものでなく、対象物を明確に限定された処理機能をさす。つまりデータと処理とを一体にして機能と考える。

ソースコードとして同一の処理機能であっても対象とするデータが異なれば別機能と考える。

これまで、一つのデータベースを処理するプログラムが複数存在する時、そのデータベースに対する一貫性制御処理(integrity control)は、それぞれのプログラム内に明示的に持たれることが多かった。プログラムの設定が個別に行われたからである。一貫性制御処理の重複は、それぞれのプログラム作成を複雑にしコスト高にするだけでなく、その重複が相互の不一致、食い違いの原因となる。一貫性制御処理を個々のプログラムから追い出し、独立に実現する考え方は一貫性独立（integrity independence）と呼ばれるものである。

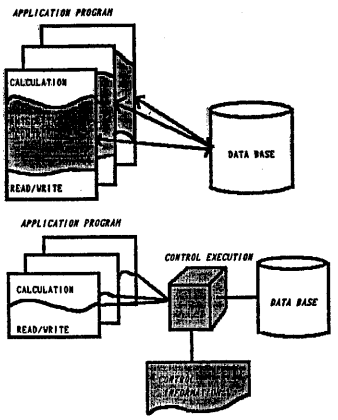


図　３　一貫性独立

ＤＯＡでは、データの内容に関与する更新プロセス（発生、追加、削除、更新）は、データ対応に決定できることから、データ毎に１箇所にまとめる。そのようにしてまとめられたプログラムは一つのデータのライフサイクルを制御するプログラムとなることからＤＬＣＰ（Data Lifecycle Control Program)と呼ぶ。

対象物を明確にしながら、対象物毎に処理を考えるアプローチはオブジェクト指向（ Object Oriented Approach)と呼ばれるものと同じである。また、対象物と共に処理を一まとまりのものと見なす考え方は抽象データ型（Abstract Data Type)のものである。データとプロセスを一まとめにすることをカプセル化（encapsulation)とも呼ぶ。カプセル化は後で述べるように、データとプロセスの相互関連を最小化するだけでなく、システムを構成するための取扱やすい、意味的にも明快なコンポーネントを得る上で有効な方法となる。

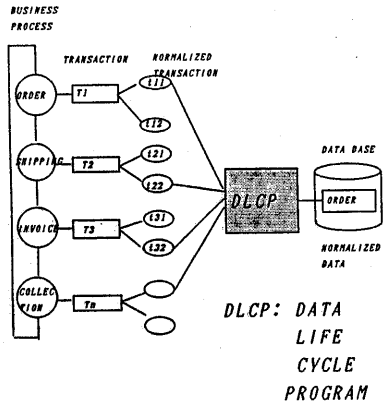


図　4　　ＤＬＣＰ

2　カプセル化とその構成原則

2．1　カプセルとは

カプセル（capsule）とは、ある働きや機能を実現する手段を一つの容器の中に封じ込めたものをいう。パッケージ(package)ともいう。その目的は認識と取扱の容易化、内部構成の保護などに置かれる。オブジェクト指向におけるオブジェクトもオブジェクトとメソッドを組み合わせたカプセルである。また、データとプロセスの両者を同時に分析し、抽象化することにもなるので抽象データ型とも呼ばれる。

カプセルは封じ込めを意味するが、特別なハード機構を必要とするものではない。データ、プロセス、および制約を１：１：１の関係に置いたものをさす。また、プロセスと制約は実現手段を問わなければ同一のものと考えることもでき、大きくはデータとプロセスの対をカプセルと考えることにする。

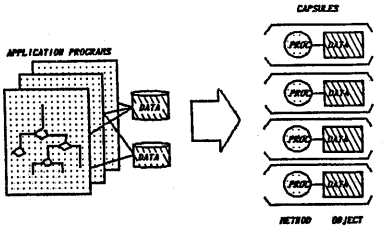


図　5　　カプセル

カプセル化の手法は、一つのシステム内で、単にデータベースとそのプロセスに限定される必要なく、図　6に示すように画面、トランザクションなどにも適用できるものである。もし、それをカプセルとして予め実現できれば、システムはそれらカプセルを実行時に結合（ダイナミックバインディング）することで作動でき、システム変更に対する融通性は飛躍的に向上できる。勿論、実行時結合のオーバヘッドは問題となる。

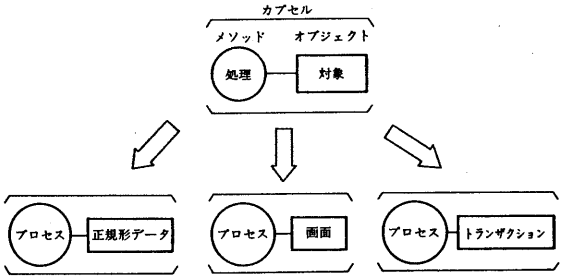


図　6　カプセルの特化

2．2　カプセル化の原則

カプセルは、単に封じ込めだけを意味するものでなく、その目的から、次のような構成要件が求められるものとなる。

（１）カプセルは一つのデータ型に従うデータ実現値とそのデータを操作するプロセス要素から構成される。
（２）カプセル内のプロセスの処理状態は有限個で固定できる。
（３）一つのカプセル内のプロセスは、たのカプセル内のデータを直接操作してはならない。
（４）カプセル内の処理は、その中に「反復」、「選択」など、自己の妥当性、正当性を保証する処理を持つべきでない。

図　７は、一つのカプセルが他のカプセルとのかかわり合いを必要とする場合を示すものである。他のカプセルに対するメッセージとしてかかわり合いを実現する。

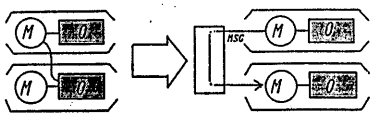


図　７　カプセルの凝集性

また図　８は、カプセル内に「選択」、あるいは「反復」処理が実現されている例を示している。選択、反復処理は当該カプセルの外部で制御できるものと言える。ただし、外部にそのカプセルを制御するコントローラが必要となる。つまり、カプセルは必ずその上位にそのカプセルを制御するコントローラを必要とする。したがって、カプセル化は「実行」とその「制御」を明確に分離することを意味する。

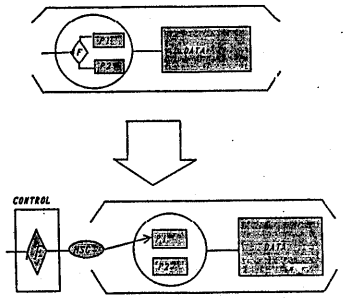


図　８　カプセルの制御

2．３　カプセルのオブジェクトクラス

カプセル内のデータはオブジェクト指向でいうオブジェクトと同じである。SmallTalk-80などに見られるオブジェクトはオブジェクトクラス（object class）をもつ。オブジェクトクラスは一つのオブジェクトのサブタイプを明快にし、その間の属性継承（inheritance of property）を行わせる。

ＤＯＡでも、オブジェクトクラスは重要な概念となる。一つのデータ型に属するデータは、少なくても正規化され明快な構造を持つ。また、汎化（generalization),特化（specialization）など抽象化操作を受ける。その操作はＤＬＣＰを設計する上で不可欠なものとなる。なぜならば、図９に示すように、一つの実体「社員」を捉えても、その中に「プログラマ」、「アナリスト」など様々なサブタイプを持つからである。サブタイプごとに固有の処理を明らかにしない限り、明快なＤＬＣＰを構成させることはできからである。

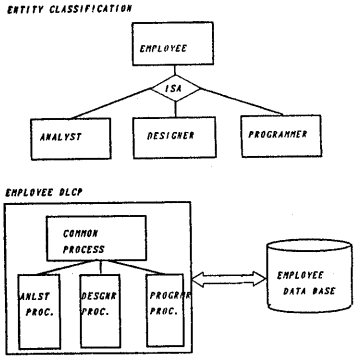


図　９　サブタイプとＤＬＣＰ

2．4　カプセルとコントロールクラス

カプセルはその内部構成が簡潔で、明快なものでなければならない。そのために、可能な限り自己の正当性、妥当性を保証する処理を自己の中に持ち込まない。先に述べたように、カプセルは必ずそのコントロールを外部に持つことを原則とすれば、そのための制御は上位のコントローラに任せることができる。

またそのような、制御と実行の分離はシステム全体におけるプロセスの重複排除に通じ、システムの単純化につながる。

図１０と図１１は、そのようなアイデアを示したものである。図１０はこれまでの伝統的アプローチを示すものである。即ち、外部からの要求（トランザクション）に対して、その要求を充足するための処理がシリアルに同期して実行される。
図のウエイターに該当するのがオンラインシステムでタスク（task）と呼ばれるものに該当する。
そのタスクはプログラムとして与えられる処理基準に従って実行される。したがって、そのプログラムはトランザクションの受付とその検証のための処理に始まり必要とされるファイル処理（料理）に関するものまで、全てを包含していなければならない。つまり、実行と制御は分離されず複雑なものとならざるを得ない。

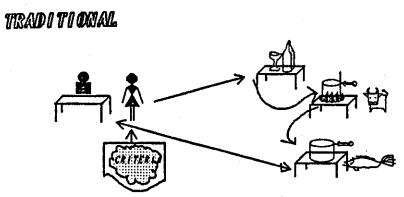


図１０　伝統的制御

これに対して、図１１はファイル処理などに専門の担当者を用意して、分業を図っているものである。料理をオブジェクト、その担当者をメソッドとしてカプセル化が実現されている。ウエイターはそれらのカプセルを制御するコントローラとなっていることがわかる。

また、それぞれの処理を規定する基準（プログラム）も分離されそれぞれのオブジェクトごとに封じ込められている。肉料理担当者は魚料理担当者の手順を気にする必要はない。手順が明快になっていることがわかる。このとき、もし「注文された料理が肉料理ならば、ワインは赤でなければならない」という命題（一貫性制約）があったとしたら、その命題は誰が責任を持つべきであろうか。肉料理担当者のものであろうか。肉料理はワインが注文されたか否かも分からない。したがって、その制約に責任を持つのはウエイターであることが分かる。

もし、ウエイターがその制約に責任を持たないとしたら、肉料理担当者も魚料理担当者も各メソッドがウエイターをカバーして、それぞれワインの色を意識しなければならなくなる。つまり制御処理が重複することになる。カプセルとその制御（コントロール）を明示的に設置することでその重複を排除できる。

オブジェクトとメソッドのカプセル化は、このように制御の明快化とその実現方法に関する明瞭な基準を提供してくれる。

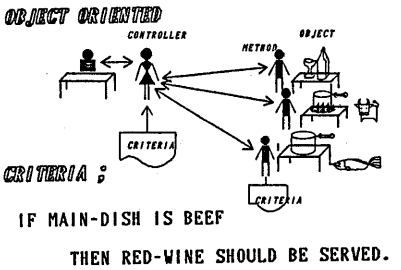


図１１　カプセルによる制御の単純化

図１１の処理構成に近いものは、既に７０年代にデータベースなどの共有資源をもつリアルタイムシステムの制御方式として実現されている。トランザクション起動方式と呼ばれるものである。具体的にはＩＢＭ社のＩＭＳのＤＣ機能などで実現されている。つまり、内部のサブタスクとして、資源対応のサブタスクを用意して、システム全体のスループッ

トを向上させるものである。データベース、システムログなどの資源をオブジェクトとしたメソッドがＤＣサブタスクであったといえる。共有資源を持つシステムの制御は、それなりに固有の方式を必要とする。

図１２は,このような考え方に従って、一般のシステムをモデル化したものである。オブジェクトとメソッドの関係は再帰的である。このような再帰的モデルでそれぞれのオブジェクト（カプセル）のレベルをオブジェクトコントロールクラス（object control class）と呼ぶことにする。

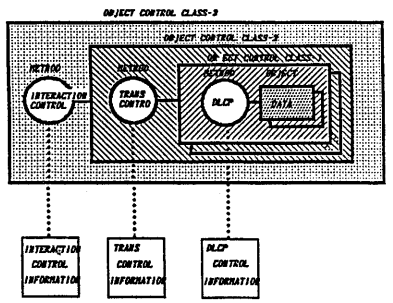


図１２　オブジェクトコントロールクラス

カプセル化されたオブジェクトの制御階層が図１２のように定義できるならば、それぞれのメソッドとしてのプロセスが果たすべき処理、特に下位のカプセルの制御、は明快になる。その制御が確実にそのレベルのメソッドによって実行される限り、下位のカプセルは一切その制御処理をその中に持つ必要がなくなる。

例えば、トランザクションを受付けるメソッドがトランザクションの妥当性、正当性の検証を確実に行えば、その下位の処理プログラム（ＤＬＣＰ）は一斉に入力検証処理から解放される。もし、
上位のメソッドによる検証が保証されない場合は
複数存在する下位のプログラムの全てが同じ検証処理を重複してその内部に持たねばならない。

プログラムから一貫性制約処理を分離する考え方は一貫性独立と呼ばれる。しかし、その実現のためには一貫性制約を抽出する方法論が不可欠である。単に、データベースのデータに関するデータ一貫性制約だけでなく、画面、トランザクションなどもオブジェクトと見なしたカプセル化が必要となる。

3　ＤＯＡシステム構成

前述のようなカプセルとコントロールクラスを前提に、システムの構成を示したものが図１３である。

データとＤＬＣＰによるカプセルが最も低位にあり、それらをオブジェクトと見なした上位のメソッドとしてトランザクション制御が存在する。さらに、トランザクション制御をメソッドとした、より大きなオブジェクトに対するメソッドとして対話制御が存在する。

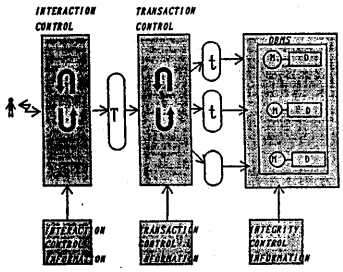


図１３　ＤＯＡシステム構成

3．１　トランザクション制御

トランザクション制御は、このレベルでなし得る制御の全てをカバーする。即ち、もしこのレベルで実施しない場合、これより低位のカプセルで重複して実施せざるを得ない制御処理をすべて実施する。具体的には次のような制御が実行される。

（１）正規形トランザクションへの分解
（２）正規形トランザクションのＤＬＣＰへの
　　引渡し制御。
（３）ＤＬＣＰの起動
（４）ＤＬＣＰ実行結果の検証

図１４は、トランザクション制御の様子を示したものである。外部から投入されるトランザクションを正規化して、正規形データをもつデータベースに対して引き渡すべきか否かを制御している。

例えば、外部トランザクションとして投入される「受注伝票」上のデータの内、データベースに反映すべきものと反映させるべきでないものを分離しなければならない。これまでこのような制御が個々のプログラム内で実施されていた。

トランザクション制御は、その制御のための情報を定義情報として参照する。その制御情報の一つはトランザクションマトリックス（図１５参照）と呼ぶもので、外部トランザクション、正規形トランザクション、ＤＬＣＰ，及び正規形データとの相互関連を示したクロスレファレンスである。もう一つは、トランンザクションの形式とトランザクション生成条件などを示したものである（図１６参照）。

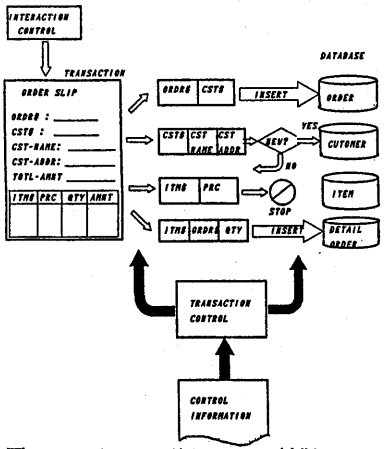


図１４　トランザクション制御

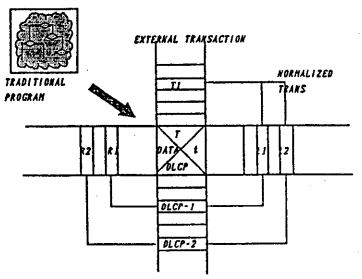


図１５　トランザクションマトリックス

```
TRANSACTION : T1

  SUB-TRAN : NAME=t11,DLCP=P1, PROCESS-TYPE=INSERT

    CREATE : UNCONDITIONAL

      FIELD : NAME=ORDR#,SIZE=10,TYPE=AN
      FIELD : NAME=CST# ,SIZE=7 ,TYPE=AN

  SUB-TRAN : NAME=t12,DLCP=P12, PROCESS-TYPE=INSERT

    CREATE : WHEN (NEW.CUTOMER) ELSE IGNORE

      FIELD :  NAME=CST#,SIZE=7, TYPE=AN
      FIELD :  NAME=CST-NAME,SIZE=30,TYPE=AL
      FIELD :  NAME=CST-ADDR,SIZE=80,TYPE=AL

  SUB-TRAN : NAME=t13,DLCP=P13, PROCESS-TYPE=INSERT

    CREATE : UNCONDITIONAL
```

図１６　トランザクション生成条件

これらの情報は、トランザクション制御がその制御を実行する上で不可欠な情報といえるが、従来の手法でプログラム内にコードとして実現されていた制御処理を、ビジブルな定義情報として抽出したことを意味する。

多くのシステム変更はこの定義情報を書き換えることで対応できるものとなる。

３．２　対話制御

トランザクション制御をメソッドとするオブジェクトを制御するクラスが対話制御クラスである。

対話制御は、外部の端末などとのメッセージの授受を行うための制御よりも、何回かのインタラクションの過程で投入されるデータの正当性に責任を持つ制御を意味する。物理的なメッセージ制御はオンラインコントロールの役割である。ここでいう対話制御は次のような制御を行う。

（１）トランザクション生成の正当性の保証
（２）対話を支援するための画面制御
（３）画面オブジェクト対応のメッソド実行

対話制御としてオブジェクトコントロールクラスが必要となるのは、トランザクションの生成を保証するためには、トランザクションが投入される段階で画面対応に生成制約を検証することが要求されるからである。もしその段階で検証しなければ、それより下位のコントロールクラスの全てのメソッドが重複してその検証を持つことになる。

図１７は、画面構成するフィールドを一つのオブジェクトと見なして、そのオブジェクトに対するメソッドとして検証制約を封じ込めた例である。対話制御はこれらの制約ていぎに従ってトランザクション生成に関する保証を行う。

このようにトランザクション生成制御のための情報も、トランザクション定義情報と同じように、定義情報として、オブジェクト対応に重複なしに保持されるので、システム変更時の対応が定義情報の書換えで行えると共に、変更箇所が複数箇所に分散しないことがわかる。

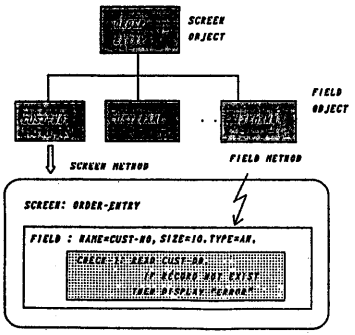


図１７　対話制御情報

４　ＤＯＡのレベルと手順

これまで、データをオブジェクトと見なしたカプセル化が、システムの単純化に寄与すると共に、プログラム言語を駆使して作るべきプログラムの絶対量そのものを削減できること述べてきた。プログラム化すべきものの多くは定義情報として実現されることも述べた。その発想はかってテーブルドリブン手法と呼ばれたものに等しい。しかし、テーブルドリブンを試みた者の多くは、テーブル相互の矛盾と食い違いによってそのアプローチを放棄せざるを得なかった。何故ならば、データそのものの正規化もせずに、データ重複を許したままでテーブル方式に臨んだからである。

今日、我々はデータ正規化の理論も、オブジェクト指向や抽象データ型などの概念も利用することができる。最も基礎となるのがデータの分析と整備に他ならないことが分かる。

したがって、ＤＯＡを実現するためには、できるだけ幅広い範囲で業務を捉えて、その中で共通データを抽出しながら、正規化を中心としたデータ標準化を図らねばならない。

データの標準化から、前述のＤＯＡシステム構成を実現するまでのレベルには大きく６段階ある［ＨＯＲＩ１、ＨＯＲＩ５］。

図１８にそのレベルを示す。この中でも最も高いハードルは第１段階である。何千、何万というデータ項目に一貫した名称を付与すること、名称で識別されるデータ項目の意味内容を定義することは、その重要さ程には受け入れられない。データの分析作業そのものが単調で、労力を要するものであることが歓迎されない理由となっている。データ分析作業の負担を削減できるデータエンジニアリングツールの実現が強く望まれる。

本稿では、データ標準化作業あるいはデータネーミングなどの方法ついては省略する。その詳細については［ＨＯＲＩ１、ＤＵＲＥ１］を参照頂きたい。

また、具体的に作業を進めるための手順に付いては図１９、２０を参考までに付与する。ＤＯＡレベル６を前提にした手順である。

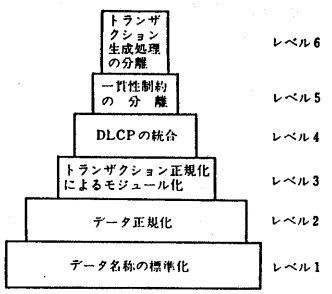


図１８　ＤＯＡのレベル

５　問題点と今後の課題

図１９、２０に示した手順は理想的なものである。現実のシステムでレベル６を達成したものはない。

技術革新を分類すれば、プロセスイノベーションとプロダクトイノベーションに分けられる。ＤＯＡも、プロセスイノベーションとしてのＤＯＡとプロダクトイノベーションとしてのＤＯＡに分けることができる。

プロセスイノベーションとしてのＤＯＡは、これまで繰り返されてきたソフトウェア開発のプロセスを、少しでも効率化することを目的にしたものである。したがって、構築すべきシステムのアーキテクチュアを変えるつもりはない。ただ、データ標準化によってデータ定義を標準化し、プログラム開発に供給することで、プログラム開発過程のトラブルを最小化しようとするものである。そのような目的のために必要とされる支援ツールがＣＡＳＥ（Computer Aided Software Engineering）として追求されている。

一方、プロダクトイノベーションとしてのＤＯＡは、開発すべきターゲットシステムの構造までをＤＯＡの発想に変えようとするものである。当然ながらドラスチックなインパクトを与えるものとなることから、容易に実現できるものとはならない。前述のカプセル化によるＤＯＡシステム構成は、そのようなイメージを示したものである。

今日、我々はプロセスイノベーションとしてのＤＯＡとプロダクトイノベーションとしてのＤＯＡの両者を手にしており、時として区別もなく論議される。語る者によってイメージが食い違うのもこの両者が混同するためと思われる。

もし、プロダクトイノベーションとしてのＤＯＡを目標にするならば、ターゲットシステムを支えるパラダイムのイノベーションも不可欠となる。カプセル化を支えるシステムソフトや一貫性独立を支えるＤＢＭＳも必要となってくる。

言うまでもなく、プロセスイノベーションとしてのＤＯＡよりも、プロダクトイノベーションとしてのＤＯＡの方が見返りも大きく、飛躍的な改善を期待することができるが、今日の技術で実現でいる範囲は限られてくる。したがって、長期的展望と短期的課題の明確な把握による実行計画シナリオの作成が重要となる。

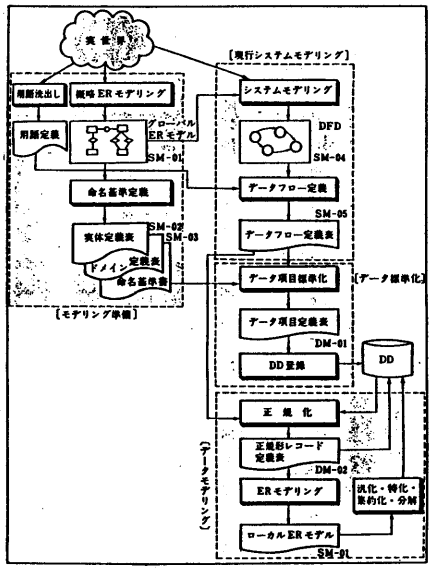


図１９　データモデリングフェーズ

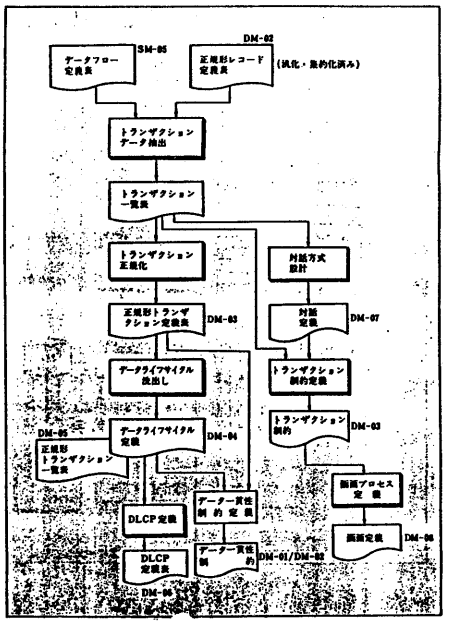


図２０　システム設計フェーズ

【参考文献】

（ＨＯＲ１１）堀内　一「データ中心システム設計」オーム社、１９８８

（ＨＯＲ１２）峠村、山田、堀内「データベースシステムの設計と開発」、オーム社、１９８３

（ＨＯＲ１３）堀内　一、山田　進、"データ中心システム設計技法"、日経コンピュータ、５／２７号、１９８５

（ＨＯＲ１４）堀内　一、"情報資源管理とデータ中心アプローチ"、情報処理学会データベース研究会、１１月１９８６

（ＨＯＲ１５）堀内　一、"データ中心設計技法の可能性と問題点"、日経コンピュータ　６／２２号（別冊）、１９８７

（ＳＡＫＡ／ＨＯＲＩ）H.Saksi,H.Horiuchi,"A Method for Behavior Modeling in Data Oriented Approach to System Design",Proc. of IEEE COMPDEC84 1984.

（ＤＵＲＥ１）W.R.Durell,「データ資源管理」ＩＲＭ研究会訳、日経マクロウヒル社、１９８７