解　説

システム開発パラダイムと高水準データモデル†

堀　内　一††

1. はじめに

近年，システム開発における CASE ツールの進展は著しく，システム開発の形態や方法論の上に大きな変革をもたらそうとしている．特に，オブジェクト指向概念とツールの登場，あるいはリポジトリとデータ中心概念の普及は，あたかもソフトウェア開発作業そのものを，プログラム開発作業からデータやオブジェクトの設計と定義作業に置き替えようとしているかにみえる．

これまでソフトウェア生産技術は，プログラムを成果物とみなし，その効率的生産を目的に，作業改善を主眼とした設計基準や手順作り，あるいは作業支援のためのツール化に注力してきたといってもよい．しかし，長期にわたるソフトウェア開発の繰り返しは，膨大なソフトウェア累積をもたらした．その累積は相互に重複を含み，たび重なる保守によりシステム全体の構造劣化を来している．そこで，要求に従ってプログラムコードを作り出すのでなく，あらかじめ資源とその操作と制約をメタ情報として用意し，そのメタ情報解釈によりソフトウェア機能を果たさせるような新しいパラダイムが求められているといえる．本稿では，そのようなパラダイムの実現手段として，CASE あるいはデータ中心，オブジェクト指向をとらえて，その方法を模索しつつ，そこにおけるデータモデルの意義について論じてみたい．

2. データモデルとその意義

2.1 データモデルとデータモデル化機能

一般に，モデル (model) とは，ある目的のために，関心をもつ対象領域あるいは対象物を，認識可能あるいは操作可能なものとするために，その本質を抽出してとらえたものである．データモデル (data model) も，ある特定な目的のために，データが表現すべき対象，あるいはデータそのものを，本質を失うことなく表現したものである．また，その表現の方法を規定する理論や規則をデータモデル化機能 (data modelling facility : DMF) と呼ぶこともある．

しかしこれまで，データモデル化機能に関する論議は，主としてデータの構造的側面についてなされてきた．データ型に関する論議やデータ項目の組合せ，あるいはレコード間の関連づけなどに関する論議である．多くのデータモデル化機能が，1960 年代前半に提案された情報代数 (An information algebra) の概念を暗黙のうちに受け継ぎ，データを「実体」，「属性」及び「属性値」でとらえる 3 元論から脱却していないからともいえる．つまり，対象としている実体の静的構造の把握がデータモデル化やデータベース化の課題であって，動的特性の実現はプログラムの課題とされてきた．たとえば，実体「社員」に関するデータをデータ項目の組（レコード）として決定するまでがデータ設計の課題であって，その値の妥当性を保証するためのデータ更新制約の検証は，そのデータを更新するプログラム開発側の課題とされた．しかし，データの意味的捕捉は，単にその構造的側面に限らず，そのデータの動的側面やデータを支配する制約側面についてもなされるべきである．B. リスコフらによる抽象データ型[2]は，主として情報隠ぺいの観点から，データと操作のカプセル化を抽象化手法の一つとして示した．データが構造的側面だけでないことを示したものといってよい．また，E. F. コッドもデータモデルを，構造部，操作部，及び一貫性部から構成されるものと定義した[3]．このような概念はクラスと継承

† High Level Data Modelling for System Development Paradigm by Hajime HORIUCHI (Institute of Advanced Business Systems, HITACHI Ltd.).

†† (株)日立製作所ビジネスシステム開発センタ

概念を取り込みながらオブジェクト指向モデルへと発展してきたといえる.

2.2 伝統的データモデル化機能の限界

代表的データモデルである関係データモデルも,値の集合を定義域とし,その定義域群から得られる直積集合を関係とみなすデータモデル化機能である.値指向(value oriented)と呼ばれるゆえんである.そのため次のような問題が,一般に指摘されている.

● 実体概念が明確でない.

実世界に存在する実体は表(関係)を構成する行(タプル)として表現される.つまり,複数の列(属性)に従った値の組が一つの実体に対応するものとなる.ある面では値そのものも実体である.一つの実体を表現するのに複数の実体を組にすることになる.言い替えれば,「社員」という名称をもつものが,列として存在するものか,表として存在するものかは一意には決まらない.また,たとえば,「親部品」と「子部品」からなる「部品」という実体を表現するには,通常,次のように表を分けなければならない.

部品 (部品番号, 部品名, サイズ)

部品構成 (親部品番号, 子部品番号, 所要量)

つまり,「部品」と「部品構成」が,それぞれ別な実体として認識されている.しかし,本来,部品の構成関係は「部品」という実体の相互関連にすぎない.「部品構成」を独立した実体とみなすことに不自然さを感じる.

● プロセス特性を表現できない.

関係データモデルは,データの構造的側面を表現できるが,データの発生,消滅,あるいは更新などのライフサイクル処理をモデルとして表現できない.一部のチェックプロセスは SQL として記述できるようになりつつあるが,データが動的側面をもっているという前提には立っていない.

● 制約を十分に表現できない.

現在,関係データベースについては SQL の機能として,参照制約,一貫性制約,あるいは整合性制約に関する論議が ISO の場でも行われている.しかし,DBMS 機能としての検討が主体であり,表現手法などに関するデータモデル化機能としての検討ではない.

● クラス概念がない.

関係データモデルには,表が表を含む,あるいは表が表に従属する概念はない.つまり,ある実体が上位のどのような集合に属するかを表現するクラス概念はない.むしろ集合を要素とする概念を拒絶してきた.

● 計算完備でない.

データモデルの問題ではないが,現在の SQL はデータベース操作に限定されており,データの演算処理は親言語に任せている.このことは,データの構造的側面だけに責任をもち,本来,データ側に一カ所で実現すべき操作,制約がプログラム側で実現されることを容認しているとも言える.

3. データ中心型手法の意義

データ中心型手法 (DOA: Data Oriented Approach) とは,システムの分析や設計に当たり,先にデータの存在を認め,それを共有資源として分析した後に,システムやソフトウェアの構造を決定する概念の総称である.

データベース設計技術の普及により,データベースの整合性と一貫性を維持しようとする概念は,要求対応にプログラムとデータを検討する個別手配形態を極度に警戒し,共有資源を設計制約として強く意識する方法論を主張するようになった.データ中心の背景には,そのようなデータ資源に対する認識の高まりがあった.

また一方ビジネスシステムの分野では,戦略情報システムなど,情報システムとビジネス活動との強固な連携が,ビジネス戦略変更にともなう既存システム変更への迅速な対処を,戦略的課題として求めている.そのため,システム変更にともなう余波を最小化し,柔軟性を確保する具体策を必要としている.

そこで,受注生産形態で進められてきたこれまでのシステム開発を,資源を前提とした多品種量産形態に移行せしめるような方策が望まれるといえる.つまり,先にデータなどの資源を認識し,それを標準化あるいはモデル化して,資源統制を組み込み,その後,資源を前提とした開発を行わせる生産形態の確立である.

4. データ中心設計の目標

データ中心設計の目的は,データの重複排除とそれに基づくソフトウェアの重複排除,さらに,

それらを通じて，できるだけ普遍的なシステム構成要素を見いだし，スリムで柔軟なソフトウェアシステムを実現することにある．その目的を達成するための手段は次のようなものとなる．

（1） データ部品化とカプセル化

データ部品とは成果物を情報とみなして，その情報成果物を効率よく生産（組立）することを目標として吟味されたデータ組（タプル）を指す．プログラムを成果物とみなすソフトウェア生産の効率化を直接目標とするものではない．むしろ，エンドユーザによる情報生産の効率化を狙うものである．

データは部品として共通化を図る上で，プログラムよりは有利である．なぜならば，共通項を繰り出すための母集団を特定しやすいからである．データはユーザの要求に従って存在するものではなく，実世界の実体またはオブジェクトに従って存在するからである．

したがって，実世界の関心領域を特定できれば，その対象領域に存在する実体や，実体の特性と振舞いの事前列挙 (pre-enumeration) も可能となり，共通化の吟味が可能となる．プログラムの部品化では，プログラム製品の母集団を特定することが難しく，かろうじて処理パターン程度のものを列挙できるに過ぎない．よく吟味された安定的なデータに，そのデータ固有のプロセスとそのデータの整合性に関する制約などを対応づけて取り扱うことを，データとプロセス及び制約のカプセル化 (encapsulation) と呼ぶ．カプセル化によりプロセスや制約の組み込みは一カ所だけですむ．カプセルは，単に情報隠ぺいの効果だけでなく，それ自体が部品としての再利用可能性をもつ．

（2） データライフサイクル処理統合

システムの複雑性を増加させる要因の一つは，データ更新処理のプログラム間分散である．従来の構造化手法では，データベース更新処理も一つの機能とみなされ，トランザクション対応にプログラム化された．その結果，データベースと更新プログラムの対応は多対多となり，システム構造を複雑化させ，プログラム変更を波及させる原因となった．

データ更新プロセスと，その更新の妥当性を保証する制約条件は，データに固有のものである．したがって，データの発生，消滅，更新のイベントを事前列挙し，そのイベント群とデータ群との対応を検討することによって，更新処理をデータ対応に統合することができる．そのようにして得られるプログラムをデータライフサイクル処理プログラム (DLCP : Data Life Cycle Program) と呼ぶ(**図-1** 参照)．DLCP はデータごとに1対1の対応となることからカプセル化が可能となる．ライフサイクル処理のカプセル化によりデータとプログラムの対応関係は単純化される．ライフサイクルを統合したカプセルは相互に高い独立性をもつものとなる．

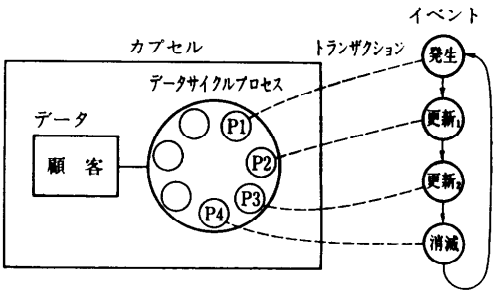


図-1 データライフサイクルプロセス

（3） 制約のカプセル化と制御独立性

ライフサイクル処理のカプセル化によりシステム構造は単純化されるが，ソフトウェア重複は排除しきれない．トランザクション処理プログラムの中にデータ更新などの処理に付帯して，入力トランザクションの検証やデータベース更新の検証処理などが組み込まれるからである．一般に，業務処理プログラムでは，そのような検証処理が平均70% ものコードを占めている．このことは，70% のコードを占める検証処理が，他の業務処理プログラムとの間で重複して組み込まれる恐れがあることを意味する．逐次処理を基本とするプログラムでは，データ更新前後の検証処理は，同期して実行される性格の処理であることから，トランザクション処理中に，IF 文による制御処理として組み込むものと考えられた．IF 文による制御処理をトランザクション対応のプログラムごとに実装することは，データ固有の制約条件をコードとして重複させるリスクに通じる．したがって，プロセス重複排除の観点からは，IF 文による制御の実装は警戒を要するものとなる．**図-2** はそのようなプログラムの例である．同一在庫データベースを更新する3本のプログラムの間に，更新制約

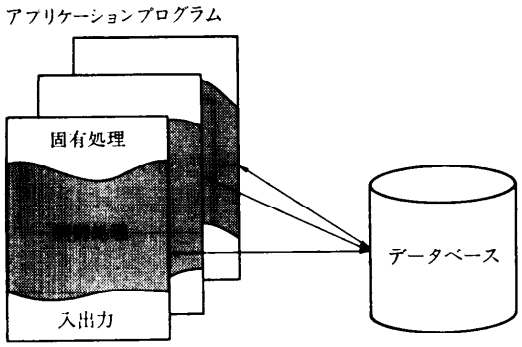


図-2 機能指向のプログラム

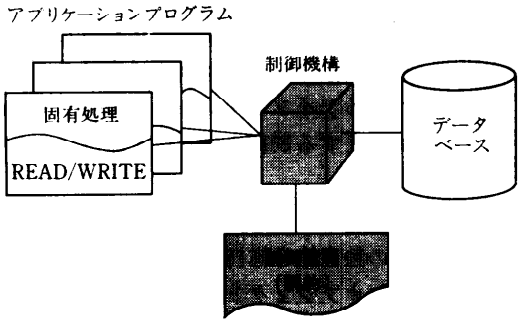


図-3 プログラムの制御独立性

として「もし安全在庫量を割り込んでいたら発注処理を行う」という処理基準を表すコードが等しく，重複して組み込まれているはずである．データベースあるいはトランザクションなどに従属している制約条件を，全てプログラム別の機能を満たす処理基準とみなしたためといえる．しかし，データまたはオブジェクト固有の制約をなんらかの型でライフサイクル処理と同様にカプセル化することで，制御処理の重複組み込みを回避できる．

カプセル化の実現手段の一つは，データと，それに対応づけたプロセスや制約に関するメタ情報を別に用意することである．データ固有の制約を，IF 文による制御処理として実装せず，外部にメタ情報として実装する方式を制御独立(integrity independence) と呼ぶことにする（図-3 参照）．

（4） データの純粋化とプロセスの単純化

必要以上に，見かけ上のプログラムコードを増やし，プログラム内部の処理フローを複雑なものとしている原因もデータに求めることができる．これまで，トランザクション対応の機能指向プログラミングでは，外部のイベントと対応付けながら処理の分岐を個別に検討する方法をとってきた．時には，例外的なイベントを識別するために，フラグやインディケータなどのデータを定義することも行われた．また，ある特定なイベントによる処理結果を次の処理に引き継ぐために，フラグなどを用いることも実務的にはよく採られた方法である．このような，プログラム内に閉じたイベント制御がプログラムを複雑にしているといえる．端末の PFK（プログラムファンクションキー）の割当などが良い例である．外部に発生したイベントを PFK の押下でプログラムに伝える方法である．PFK をどのようなイベントに割り当てるかは，業務上の外見的認識に基づくことが多い．時には，プログラム側からフラグ代わりに PFK 割当を求めることもある．そのような方法は，イベント発生要因への認識と分析を欠如させるものとなる．たとえば，顧客管理業務で PFK を次のように割り当てたとする．

【例 1】 PFK 1:「女性で会員の場合」
PFK 2:「女性で非会員の場合」
PFK 3:「男性で会員の場合」
PFK 4:「男性で非会員の場合」

このような PFK 割当は，必要 PFK 数を多くするだけでなく，これを受けるプログラム側の CASE 文の分岐も多くする．もし，業務として新たに「準会員」を導入したら，どのような変更が発生するかは想像するまでもない．PFK 割当は全面見直しとなり，その見直しが何本のプログラム変更を引き起こすか分からない．これに対して，PFK を次のように割り当てれば問題は緩ぐであろう．

【例 2】 PFK 1: 女性
PFK 2: 男性
PFK 3: 会員
PFK 4: 非会員

つまり，「性別」と「会員」の 2 種類の実体が，それぞれ，「女性」及び「男性」あるいは「会員」及び「非会員」というサブタイプをもつに過ぎない．

仮に，プログラム内で CASE 文や IF 文の使用を許したとしても，サブタイプ識別に用いれば，その分岐は単純化され，変更も容易となる．【例 1】に示す PFK データは「多事実 (multi-fact) データ」[7] などと呼ばれるものである．正規化手法では発見できない．COBOL プログラムな

どで REDEFINE 句を用いて再定義するデータもその代表例である．一つのデータ項目が表現する意味を単一なものとする分析作業を純粋化とも呼ぶ．

5. オブジェクト指向モデル化機能

これまで述べてきたデータ中心アプローチからの，データモデル化機能への要請を満たすものの一つがオブジェクト指向のモデル化機能である．その要件は次のようなものとなる．

5.1 オブジェクト指向モデルの要件

（1） 普遍的データ概念としてのオブジェクト

モデル化機能には，単にデータベースのデータだけモデル化の対象とするのでなく，ソフトウェアが対象とする画面，帳票，トランザクションなど全てについて，それぞれのデータ概念における個別性あるいは特殊性を消去して，抽象的なオブジェクトとして統一した概念で取り扱えることが求められる．

これまでも，データの設計において重要な課題の一つは，個別の立場からのデータへの認識を消去して，できるだけ普遍的なデータを設計することにあった．その普遍性は，ビューとしてのデータ要素の組合せの上にだけ求めるのでなく，データ要素そのものの認識に対しても求められるべきである．たとえば，カラム（列）の集合をテーブル（表）として認識するデータ概念は関係データモデル固有のものである．その概念に従って操作を考えることは，データ要素の認識に関係データモデル固有概念の拘束を受けるものとなる．たとえば，表操作とカラム操作は別なインタフェースを用いなければならない．また，テーブルにおけるカラムは，実体の属性とみなされてきた．そのため属性に固有のライフサイクルプロセスなどを想定することが難しかった．しかし，カラムもオブジェクトとみなせれば，カラム固有のライフサイクル処理を定義できるものとなる．

（2） カプセルとしてのオブジェクト

オブジェクトは，その内部にデータをもち，そのデータ固有の操作と固有の制約をカプセル化したものを指す（**図-4** 参照）．カプセルは，そのデータ側面に着目すればデータ部品とみなすことができ，プロセス側面に着目すればプログラム部品ともみなせる．カプセルをシステムにおける有意な

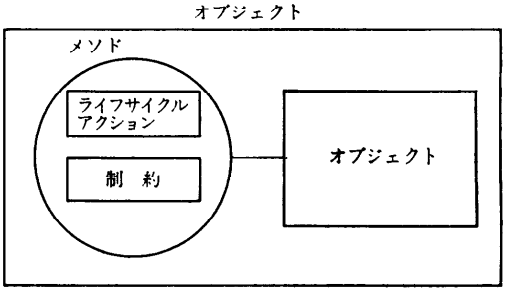


図-4 カプセルとしてのオブジェクト

要素とみなし，部品として利用するためには，カプセル相互の独立性を確保する方法をもたねばならない．その一つは，カプセル内に当該のオブジェクトに関するライフサイクル処理と当該オブジェクトを対象とする制約命題だけを組み込むことである．

（3） オブジェクトの汎化/集約化サポート

カプセルの独立性を維持するためには，カプセル自体，別なオブジェクトを内包する集約化機能が必要となる．つまり，オブジェクトはオブジェクトを含むことができる再帰的な関係をも定義できなければならない．一つの画面は，複数のウィンドウやアイコンから構成され，それぞれもまた複数の画面フィールドから構成される．あるいは，一つのデータベース(表)は複数のカラム(列)から構成される．それらの関連は，画面，フィールド，表，あるいは列といった相互独立なオブジェクトの上に定義されるべきであり，それぞれのオブジェクトに対して制約やライフサイクル操作を定義できなければならない．つまり，画面オブジェクトの制約や操作とフィールドオブジェクトの制約と操作は異なるからである．そのために，カラムと表の関係，あるいは画面と画面フィールドの関係などを，オブジェクト間の集約関係として定義できることと，その集約関係にそって求められる集約オブジェクトを実現する機構が必要となる．また，画面も「受注画面」と「受注変更画面」は別々なオブジェクトでありながら，多くのフィールドを共有する．したがって，それらのオブジェクトを汎化階層でとらえ，共通項を継承させる手段は不可欠なものとなる．

5.2 オブジェクトとメタ情報

（1） メタ情報とは

システム構成要素を，画面やデータベースなど

個別データ概念でなくオブジェクトとして統一的に操作可能とするためには，個別データ概念による見方（ビュー）と普遍的オブジェクトの記述とを分離し，両者のマッピングを実現する必要がある．そのマッピングのためには個別のデータ概念と普遍的なデータ概念とを記述したメタ情報が必要となる．

メタ情報とは，操作の対象を表現したスキーマ情報の集まりを指す．これまでメタ情報はメタデータとしてデータベース中のデータについてのみその形式側面を記述していた傾向が強かったが，普遍的オブジェクトを記述するスキーマをメタ情報とすることで，画面トランザクションなども等しくとらえることが可能となる．換言すれば，オブジェクトモデルによる抽象化は，異なる個別データ概念でとらえられていたシステム要素をメタ情報化する上できわめて都合良いものとなる．

メタ情報は，システムを生成したり，稼働させる上で，その整合性や一貫性を維持する上で有効な情報となる．システムの設計図に該当するものであり，メタ情報の精度がシステムの精度を決定するものとなる．

（2） メタ情報のレベル

オブジェクトを表現するメタ情報は，まずメタオブジェクトとメタメタオブジェクトに分けられる．このような多階層のメタ情報の概念は，ISO/IEC JTC 1 SC 21 で検討中の IRDS（情報資源辞書システム）に示されている[4]．

メタオブジェクト（meta object）は，実世界または実システムに存在するオブジェクトを実現値として，そのタイプ（型）を記述したスキーマ情報である．つまり実世界または実システムに存在するオブジェクトの構成を示すものである．

メタメタオブジェクト（meta-meta object）は，メタオブジェクトの構成を示すメタ情報である．つまり，オブジェクトのメタ情報そのものを実現値として，そのタイプ（型）を記述したスキーマ情報である．**図-5** はそのようなメタ情報をバックマン線図で示したものである．それぞれの長方形は表（テーブル）を表す．ただし，ここではメタ情報を示すのに関係データモデルによるデータ概念を使うことにする．

（3） オブジェクトの視点

システム開発において，メタ情報には二つの異なる視点のオブジェクトが必要となる．一つの視点は，実世界をオブジェクトとしてとらえるもので，ビジネスオブジェクトモデルと呼ぶことにする．もう一つの視点は，ソフトウェアとして実現されるシステムをオブジェクトとしてとらえるものでシ

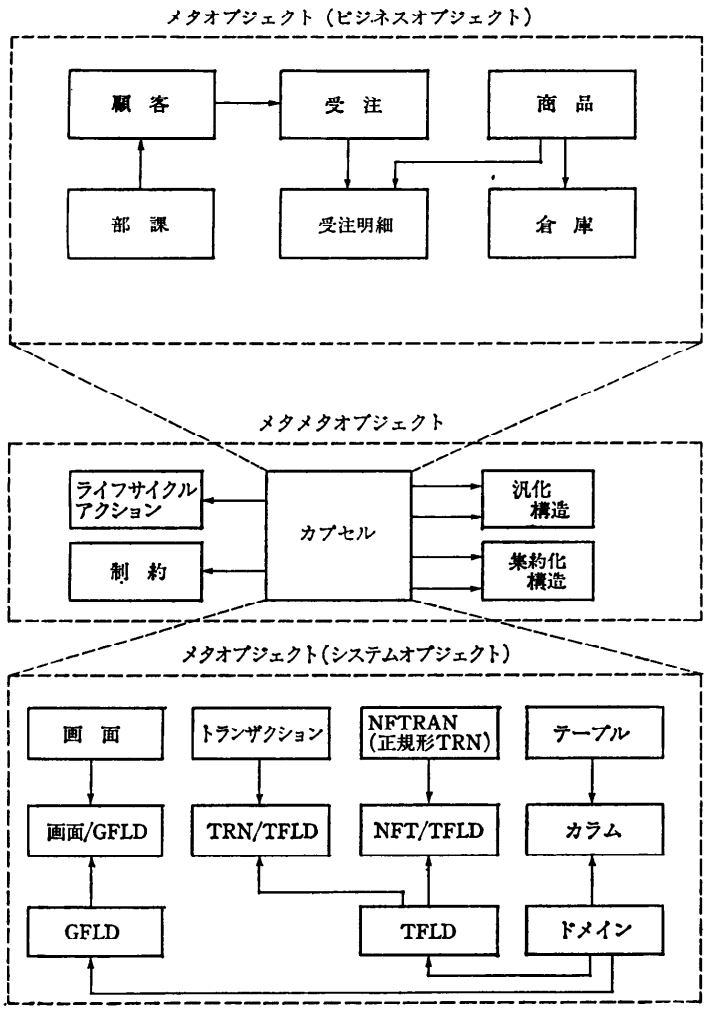


図-5 メタ情報のレベル

ステムオブジェクトモデルと呼ぶことにする．ビジネスオブジェクトもシステムオブジェクトも，それぞれ固有のライフサイクルあるいは制約を定義でき，オブジェクトとしての取扱いは統一されたものとなる．

5.3 オブジェクトメタ情報のサンプル

システムやソフトウェアの設計方法を述べるには，手順や個々の作業の基準を述べるよりも，それらの結果として得られる情報を形式化されたメタ情報として表現するほうが効率的である．前述のメタ情報のサンプルを示すことで，設計過程に必要となる作業や基準の記述に替えることにする．

（1） ビジネスオブジェクト

実世界における関心オブジェクトとその対応を示すものである．**図-6** にバックマン線図によるオブジェクト表現の例を示す．「顧客」や「受注」がオブジェクトとして認識されている．「個人」と「法人」のサブタイプに分類される「顧客」オブジェクトの汎化階層はメタメタオブジェクトの「汎化」表に表現される．

また，個々のビジネスオブジェクトのライフサイクルは，追加，削除，更新のプロセスとしてメタメタオブジェクトの「アクション」表に登録される．さらに，それらのライフサイクル遷移の過程で検証されるべき制約条件はメタメタオブジェクトの「制約」表に登録される．

（2） システムオブジェクト

情報システムを構成するシステム要素をオブジェクトとするものである．表，列，ドメインなど関係データモデル固有のデータ概念は汎化されたメタメタオブジェクトの実現値となっている．画面やトランザクションも同様である．表，列など，各オブジェクトはライフサイクルとして，追加，削除，更新のアクションをもち，それらはメタメタオブジェクトの「アクション」表に登録される．オブジェクトの制約も同様である．**表-1** は「表」オブジェクト，**表-2** は「カラム」オブジェクト，さらに**表-3** は「ドメイン」オブジェクトの例を示すものである．

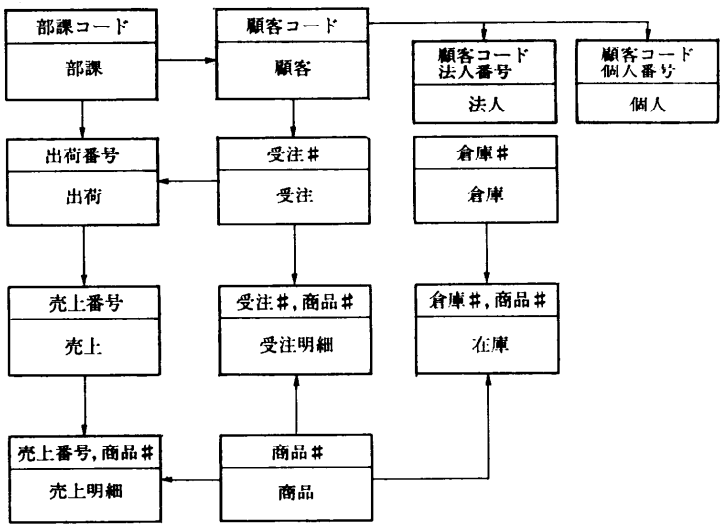


図-6 ビジネスオブジェクトモデル

（3） 制約の登録

オブジェクト固有の制約は，形式制約（データ型など），存在制約（追加，削除の規則），値制約（値の範囲，妥当性を検証する規則）などに分けら

表-1 表オブジェクト

オブジェクト番号	オブジェクト名	主キー
RT 01	顧客表	CT 01
RT 02	受注表	CT 10
RT 03	部課長	CT 20
⋮	⋮	⋮

表-2 カラムオブジェクト

オブジェクト番号	オブジェクト名	表オブジェクト	ドメイン
CT 01	顧客番号	RT 01	DM 01
CT 02	顧客名	RT 01	DM 02
CT 03	顧客住所	RT 01	DM 04
⋮			
CT 10	受注番号	RT 02	DM 01
CT 11	受注顧客番号	RT 02	DM 01
CT 12	受注日付	RT 02	DM 05
⋮	⋮		

表-3 ドメインオブジェクト

オブジェクト番号	オブジェクト名
DM 01	識別子
DM 02	法人名
DM 03	個人名
DM 04	地名
DM 05	日付（和暦）

れる．制約は条件式などの論理的記述で表現されるものである．時には，コードとして作り込まれることもある．**表-4** はシステムオブジェクトのカラム制約，ドメイン制約を示すものである．

表-4 制 約 表

オブジェクト番号	オブジェクト名	形式制約	存在制約	値制約
CT 12	受注日付	DM 05 と同じ		TT="平成" YY<=03
DM 05	日付（和暦）	TT:YY/MM/DD, not null,		if TT="昭和" :YY=[01, 64], if TT="平成" :YY=[01, 03].
RT 01	顧客表		【追加】 CT 01 が重複しないこと 【削除】 全ての受注明細の入金完了	

制約を表やカラムなどのデータベース側のオブジェクトだけでなくトランザクションや画面，及びそれらを構成するフィールドをオブジェクトとみなし，それぞれに制約を定義することが，これまでプログラムでIF 文を用いて記述されていた制御処理を，プログラムから排除し，先に述べた制御独立性を実現する方法となる．しかし，ここで問題となることは，オブジェクトの相互関連に関する制約である．カラム間の関係あるいはフィールド間の関係など複数のオブジェクトの上に定義される制約である．たとえば，先にみた「在庫残が安全在庫量を割り込んだら発注せよ」という制約は，オブジェクトとして「在庫残」，「安全在庫量」のいずれにも定義できない．なぜなら，いずれか一方のオブジェクトにカプセル化すれば，他方との関連を生じ相互に依存関係を発生させるからである．そのようなとき，「在庫残」と「安全在庫量」から構成される集約オブジェクト「在庫更新」を定義し，その集約オブジェクトに，先の制約を定義することでオブジェクトとしての相互依存性は除去できる．ただし，集約オブジェクト内の「在庫残」及び「安全在庫量」の参照と更新は「在庫更新」オブジェクトの制約とアクションで制御されるものとなる（**図-7** 参照）．

（4） メタ情報主導型システムと CASE

リポジトリ（情報資源貯蔵庫）に，図-5 のよう

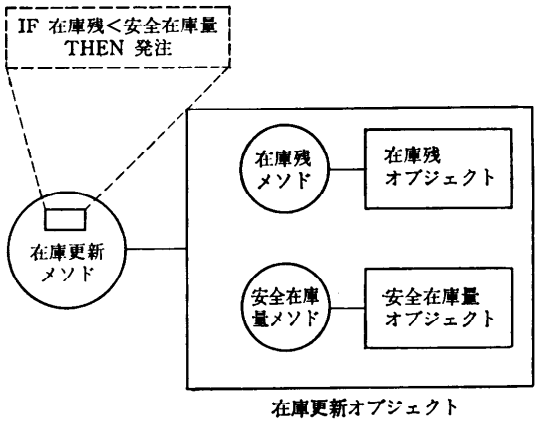


図-7 集約オブジェクトによる制約カプセル化

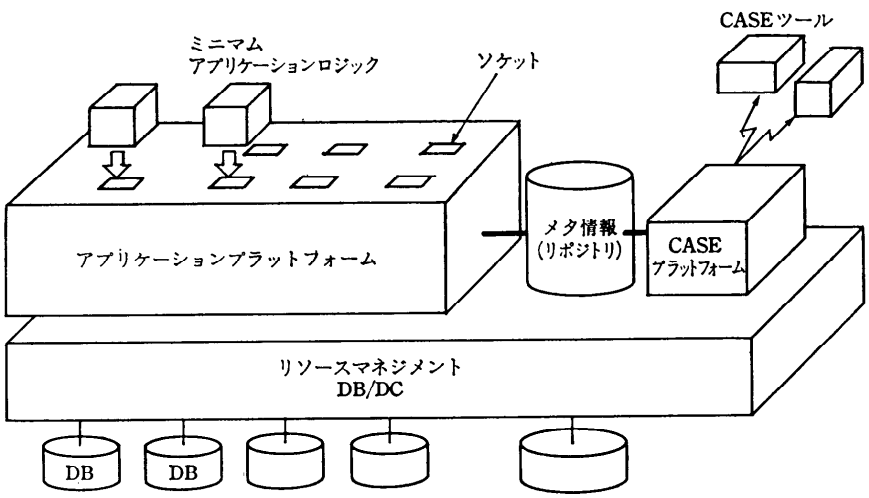


図-8 メタ情報とプラットフォーム

なオブジェクトメタ情報として表現されるシステムの実現方法には，大きく2通りある．一つはメタ情報によりプログラムを生成する方法である．もう一つはメタ情報を，直接参照しながらメッセージ授受によってカプセルを稼働させるものである．前者は今日の CASE ツールの多くが目標としている方式である．後者はプログラム生成はなく，メタ情報の変更も再結合を必要としない．図-8 は，そのような仕掛のイメージを示すものである．オブジェクトの開発用の CASE ツール群を支援するミドルソフトが CASE プラットフォームである．オブジェクトの実行制御を負担するものがアプリケーションプラットフォームである．アプリケーションプラットフォームとリポジトリによって，これまで多大のプログラムを費やして実現されてきたシステムは，重複をもたないオブジェクトと，メタ情報に置き換えられるものとなる．

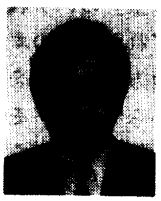
6. おわりに

オブジェクトモデル化において，最も大きな障害はカプセル化すべき事項の列挙網羅性となる．しかし，今日，多くの企業がもつ膨大なソフトウェアの中に，多くのビジネスルールとデータは埋め込まれているはずである．既存システムの再利用を目的に，既存プログラムから論理を抽出するリバースエンジニアリングは，オブジェクト分析のための手段とも考えることができる．さらに，ラピッドプロトタイピングとして知られる手法も，いったん，要求対応にソフトウェアを開発してから，そのソフトウェアを解析して，データやルールを抽出し，オブジェクトとして再構成するための手段と考えられる．

参考文献

1) 堀内 一：データ中心システム設計，オーム社 (1988).
2) Liskov, B. et al.: Specification Techniques for Data Abstraction, IEEE Trans. Software Eng., SE 1, 1 (1975).
3) Codd, E. F.: Relational Database: A Practical Foundation for Productivity, CACM, Vol. 25, No. 2 (1982).
4) Information Resource Dictionary System, Service Interface, Working Draft, ISO/IEC SC 21/N 4895 (1990).
5) Sakai, H. and Horiuchi, H.: A Method for Behavior Modeling in Data Oriented Approach to System Design, IEEE COMPDEC (1984).
6) Shlaer, S. and Mellor, S. J.: Object-Oriented System Analysis, Yourdon Press (1988).
7) Tanker, D.: Fourth Generation Data, Prentice Hall (1989).
8) 酒井博敬，堀内 一：オブジェクト指向入門，オーム社 (1989).
9) 穂鷹良介：データベースシステムとデータモデル，オーム社 (1989).

(平成3年3月25日受付)



堀内 一（正会員）

昭和43年早稲田大学商学部卒業．同年(株)日立製作所入社．(株)日立製作所コンピュータ事業部及びビジネスシステム開発センタを兼務．教育本部で社内外のデータベース教育及びコンサルテーションなどを経て，現在はビジネスシステム開発センタで CASE コンセプト開発及びデータ中心設計技法のコンサルテーションに従事．同社のシステム開発技法 HIPACE, HIPLAN の開発にも従事した．研究テーマ：データモデリング及びデータ中心システム設計技法など．著書：「データ中心システム設計」，「オブジェクト指向入門」，「データベースシステムの設計と開発」など．情報処理学会情報規格調査会「概念データモデル化機能専門委員会」幹事，日本規格協会「情報資源スキーマ調査研究委員会」幹事．システム監査学会，経営情報学会各会員．