

データ中心分析によるプログラム論理の抽出

山川 敦夫 宮本 信夫 上林 高治 石本 真希
株式会社 オージス総研

秋庭 真一 山村 信幸 堀内 一
株式会社 日立製作所

リバースエンジニアリングにおいて、ソースコードをオブジェクトに変換・整理することにより、プログラムから業務ルールを抽出する方法について述べる。リバースエンジニアリングを既存のプログラムから、再利用可能な要素を抽出する過程と位置づける。その抽出作業においては、データ中心分析の考え方にに基づき、ソースコード内のデータとそれらに対する制約・操作を関係づけ、カプセルとしてのオブジェクトに変換した。次に、カプセルから業務ルールの抽出および意味解釈を試みた。ここで述べる方法により、COBOLソースコードをオブジェクトに変換・整理する方法をほぼ確立できた。

Extraction of Program Logic according to Data Oriented Analysis

Atsuo Yamakawa Nobuo Miyamoto Takaharu Kamibayashi Maki Ishimoto
Osaka Gas Information System Research Institute Co., Ltd.

Shinichi Akiba Nobuyuki Yamamura Hajime Horiuchi
Hitachi, Ltd.

In the process of reverse engineering, the method of extracting the business rules from source codes by converting the source codes into objects and adjusting them in the model are discussed. Reverse engineering is regarded as the process for extracting or selecting reusable components from existing programs. In that extracting process, according to Data Oriented Analysis, data and their constraints and/or actions are related each other, and they are converted into objects as capsule. After that, we tried to extract and interpret the business rules from the capsules. The method to convert the source code into objects and to adjust them in the model is almost established by the method described here.

1 はじめに

保守性が高く、変更に対する柔軟性をもつソフトウェアを開発するためには、プログラム間に重複する恐れのある要素をメタ情報としてリポジトリ内に一元管理し、共有することが不可欠である。そこで、既存のプログラム群から、それら重複する恐れのある要素を抽出し、再利用可能なものに整理する方法として、リバースエンジニアリングが重要となる。

本研究では、データ中心アプローチ¹⁾をベースとする。すなわち、リバース過程は、既存のプログラムからコードで表現されているデータと操作を抽出し、オブジェクトへと変換する過程と考える。リバース過程は、次に示す3種の過程から構成される。

(1) データリバース

データに関する情報(名称、タイプ、桁数など)をプログラムから抽出し、そのデータの持つ意味を明確にする過程。すなわち、データに関する情報を基に、そのデータが表す実世界での実体を明らかにする。

(2) プロセスリバース

それぞれのデータに固有のライフサイクルプロセス(LCP)を明らかにする過程。LCPとは、データの生成、変更、消滅に関するプロセスを指す。ただし、そのプロセスは前後に、さまざまなチェック処理などを持っている。これらのチェック処理は次に述べるルールリバースの課題である。したがって、プロセスリバースの課題は、データの生成、変更、消滅に関する処理がどの時点で行われるかを明らかにすることである。

(3) ルールリバース

プログラム内にコードとして埋め込まれている判定処理やチェック処理などの制約を分解して、データごとに固有の制約としてカプセル化する過程。複数のオブジェクトに対して固有の制約は、複数のオブジェクトによって構成される集約オブジェクトの制約として定義する。プログラムを構成する多くのコードは、このような集約オブジェクトの制約として置きかわる。

本論文では、上記のリバース過程の内、ルールリバースについてその詳細を述べる。特に、ソースコード内のデータとプロセスをオブジェクトとして変換し、カプセル化する方法、およびカプセル化の結果、さらにはそれらカプセルからの業務ルールの抽出の試行例について報告する。

2 オブジェクトによる既存プログラムのモデル化の概念

リバースエンジニアリングの対象となるものは、計算機で実行されるプログラムである。リバースエンジニアリングは、このようなプログラム内のソースコードから再利用を目的としたオブジェクトを導こうとするものである。再利用には、次に述べるような条件をすべて満たすオブジェクトを抽出することが不可欠となる。

- ・プログラム内の特定の制御処理に従属しないコードの結合であること。
- ・特定のシステムの実行環境に依存しないコードの結合であること。
- ・特定のシステム実現方式に依存しないコードの結合であること。

これらの条件を満たすコードの結合群を導くためには、従来からのプログラム概念を離れ、新たな枠組み・概念を導入しなければならない。新たな枠組みには次のような要件が求められる。

- ・変化し得る要素に対する透過性(制御処理、環境、実現方式に対する)を保証するもの。
- ・外部実世界における、存在、分類体系に従い、共通の認識を可能とするもの。
- ・コードの結合群相互の独立性(相互に依存関係を持たないこと)を保証するもの。

しかしながら、再利用を可能とする上で必要となる全ての情報が、コードによって表現されているとは限らない。そこで、本研究では、リバースおよびフォワードの両過程で用いるソフトウェアのモデルを大きく概念モデルと物理モデルに分け、再利用情報の体系化を図ることにした。

(1) 物理システムオブジェクトモデル

プログラムとして実現されているオブジェクトを表現するモデル。具体的には、ソースコード内に定義されているデータと処理プロセスである。データとプロセスは、カプセルとしてオブジェクトに変換・整理される。したがって、物理システムオブジェクトモデルは、ソースコードで構成されているプログラムを最も忠実に表現したモデルである。

(2) 概念システムオブジェクトモデル

プログラムは、特定のビジネス領域での業務処理を計算機で実行させるためのものである。プログラムが実世界のビジネスを忠実に反映しているわけではないが、プログラムが対象としているデータは、ビジネス領域のオブジェクトを表現したものと、プログラム内部またはプログラム稼働環境におけるオブジェクトを表現したものとに分けら

れる。前者には、データベースの項目や、画面、帳票の項目などがあり、後者には、フラグ、リターンコード、カウンタなどがある。

概念システムオブジェクトモデルは、前者のビジネス領域のデータだけをオブジェクトとして抽出し、モデル化したものである。

(3) ビジネスオブジェクトモデル

概念システムオブジェクトをベースに、ビジネス領域における実体およびそれらの関係をモデル化したものである。ビジネス領域を表現したオブジェクトである点では、概念システムオブジェクトモデルと同じであるが、ソースコードとして表現された制約や操作を、人間の理解できる表現に翻訳し、オブジェクトとしてモデル化したものである。

これらのモデル概念は、リバース結果を受け止めるリポジトリのメタ情報構造に反映される。図2-1は、上記3種類のモデルの関連を示すものである。これらのモデルをリポジトリに格納することで、モデルの関連付けが可能となり、物理的システムオブジェクトモデルにカプセル化されたコードの再利用可能性を高めることができる。

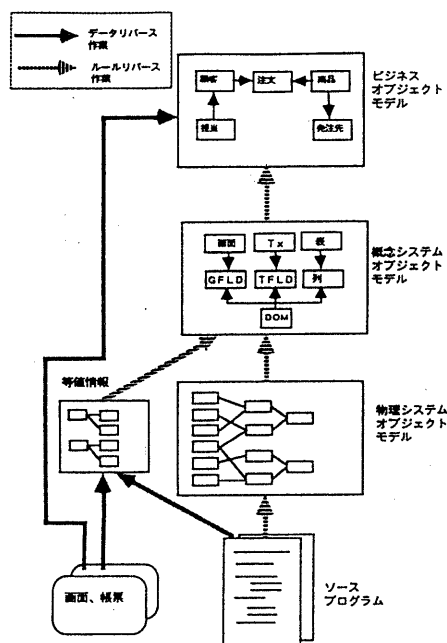


図2-1 リバースエンジニアリング用モデル

3 物理システムオブジェクトの抽出

物理システムオブジェクトとは、前述の通り既存プログラムの中に埋め込まれたデータとプロセスを表現するオブジェクトである。すなわち、ソースコードによって表現されているデータとプロセスを、カプセルとしてのオブジェクトに置き直したものである。ソースコードから抽出されるオブジェクトは、次のように分類することができる。

3.1 データオブジェクト

プログラム内で使用されるデータを指す。次のようなものから構成される。

- ・ファイルまたはデータベース中のデータ
- ・画面中のフィールドに対応したデータ
- ・トランザクションフィールドに対応したデータ
- ・プログラム内に独自に定義されたデータ

これらのデータオブジェクトは、それぞれ次のような役割をもつ。

- ・実世界オブジェクトの状態表現
- ・業務イベントに対応した外部オペレーションの表現
- ・プログラム内のプロセスの制御

また、これらのオブジェクトは、いずれもドメインを核としたオブジェクトからなる集約オブジェクトとして定義されるものである。

3.2 プロセスオブジェクト

プログラムとして実行すべきプロセスを表現したオブジェクト。次のように分類される。

(1) 2項オブジェクト：基本的データ操作を表現するオブジェクト。データ操作を、1対の(2つの)データに対するものとして捉え、その操作と1対のデータから構成される集約オブジェクトと見なしたものの。

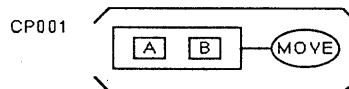
例えば、

MOVE A TO B.

として表現されるプロセスは、オブジェクト識別子をCP001とすると

CP001[(A, B) MOVE]

または、



と表現される。プログラムの基本プロセス(データ移送、演算、論理演算、など)は、まずこの2項オブジェクトとして表現される。

(2) 制御オブジェクト：2項オブジェクトとして表現されたプロセスに対する制御を表現したオブジェクト。さらに次のように分類できる。

- ・制約の表現のための制御オブジェクト
- ・制約の検証と操作の実行指示のための制御オブジェクト

一般に、制御はプログラム内で、IF～THEN～の形式をとることが多い。しかし、必ずしもIF文だけで制御が表現されているわけではなく、EVALUATE文やAT END文でも表現される。制御オブジェクトは、命題を表現する2項オブジェクトと、その判定結果に基づく操作を表現する2項オブジェクトの集約から構成される。図3-1はそれらの関係を示したものである。

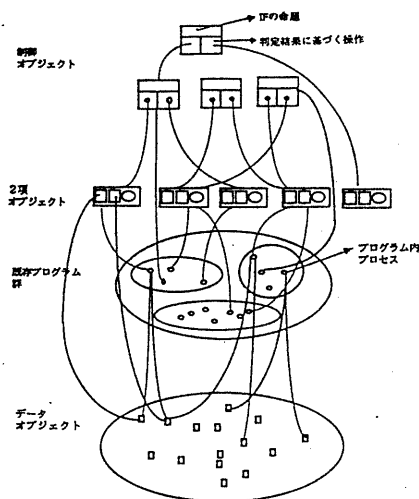


図3-1 各種オブジェクトの関係

4 2項オブジェクトによるソースコードの表現

ここでは、ソースコードによって表現されているプロセスを、カプセルとしてオブジェクトに置き直す方法を示す。カプセルは、3章で述べた2項オブジェクトを基本として成り立つ。例として、次に示すソースコードを用いる。また、オブジェクトに変換、すなわち、カプセル化した結果を表4-1に示す。

```
M3-HEIKIN.
IF W-SAVECLASS = INCLASS
  ADD 1          TO W-CLASSNINZU
  ADD INSEITOTEN TO W-CLASSTEN
```

ELSE

PERFORM M4-OUTPUT.

M4-OUTPUT.

MOVE W-SAVECLASS TO OUTCLASS.

COMPUTE OUTCLASSHEIKIN =

W-CLASSTEN / W-CLASSNINZU.

WRITE OUTDATA FROM W-OUTDATA.

表4-1 オブジェクト変換結果 (カプセル化の結果)

カプセルNo.	カプセル構成要素	メソッド
CP001	CP002	
CP002	CP004	
CP002	CP005	
CP002	CP006	
CP002	CP007	
CP003	CP008	
CP003	CP010	
CP003	CP011	
CP004	W-SAVECLASS	=
CP004	INCLASS	=
CP005	1	ADD
CP005	W-CLASSNINZU	ADD
CP006	INSEITOTEN	ADD
CP006	W-CLASSTEN	ADD
CP007	CP003	PERFORM
CP008	W-SAVECLASS	MOVE
CP008	OUTCLASS	MOVE
CP009	W-CLASSTEN	/
CP009	W-CLASSNINZU	/
CP010	OUTCLASSHEIKIN	COMPUTE
CP010	CP009	COMPUTE
CP011	OUTDATA	WRITE
CP011	W-OUTDATA	WRITE

この例は、COBOLステートメントに現われる代表的なものを記述してあるが、以下に各ステートメント別のカプセル化の方法を示す。

(1) MOVE、ADD

これらのステートメントの対象となるオペランドは2つであり、そのまま2項オブジェクトに整理できる。

(2) COMPUTE

右辺のオペランド、すなわち、演算に使用される変数は、通常2つ以上ある。A=B+C*Dという演算式があった場合、C、D、*でまず2項オブジェクト化し、つぎにB、CとDからなる2項オブジェクト、+で2項オブジェクト化する。最後に、左辺と右辺とで2項オブジェクト化する。

(3) IF文

IF文は、[命題部、命題が真の時の操作、命題が偽の時の操作]という構造から成っている。また、命題が真または偽の時の操作は、複数のステートメントから成り立つこともある。したがって、IF文を表すカプセルは、これら複数のカプセルを集約した構造で表現する。なお、命題については、88レベルのデータ項目を用いた条件文を除いては、[命題オペランド、比較演算子、命題オペランド]または[条件式、論理演算子、条件式]の構造をとるため、2項オブジェクトによる表現が可能である。

(4) PERFORM文

PERFORM文では、セクション名またはパラグラフ名をカプセル構成要素とし、PERFORMがメソッドとなるカプセルで表現する。また、各セクションやパラグラフ内の複数のステートメントは、それらを集約した1つのカプセルとして表す。

上記に示すように、データ操作に対するステートメントは、2項オブジェクトによる表現が可能であるが、一方PERFORMやSTOP RUNといったプログラム制御に関するステートメントは、2項でのオブジェクト表現にならない場合もある。

これらのカプセルを格納するためのモデルをプロセス／ルール整理モデルと呼ぶが、その構造を図4-1に示す。このモデルは、前述の物理システムオブジェクトモデルの一部をなすもので、ソースコードのプロセス部分を表すものである。このモデルを構成する5つのテーブルの内容は、次の通りである。

(1) カプセル表

ソースコードの全プロセスに対するカプセルを格納したテーブル。それらを識別するためのカプセルNo.およびカプセル名を持つ。

(2) カプセル構成表

カプセルの構成内容を表すテーブル。

各カプセルを構成するオブジェクト（カプセル構成要素）およびそれに対するメソッドを表す。

(3) IF文表

IF文に対応するカプセルのみを格納したテーブル。各IF文を識別するためのIF文No.とIF文全体を表すカプセルのNo.を持つ。

(4) 命題表

IF文の命題を構成する各条件式に対するカプセルのみを格納したテーブル。その条件式が属するIF文のNo.と条件式を表すカプセルのNo.を持つ。

(5) アクション表

IF文による命題判定後のアクション（操作）を表すテーブル。IF文の命題が真の場合の操作に対するカプセル、また偽の場合の操作に対するカプセル、およびそれらの真偽の区別を示す。

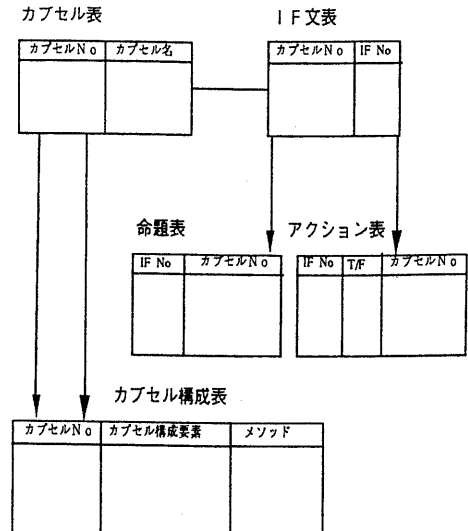


図4-1 プロセス／ルール整理モデル

5 カプセル化の実例

ここでは、実際の既存プログラムのソースコードの全プロセスをカプセル化する場合の手順・結果について述べる。対象としたソースコードは、5本のCOBOLプログラムであり、その概要を表5-1に示す。

表5-1 試行用COBOLプログラムの概要

プログラム名	プロシジャ部 ステップ数	プロシジャ部以外の ステップ数	総ステップ数
PGM-1	187	404	591
PGM-2	406	3440	3846
PGM-3	1122	5470	6592
PGM-4	1692	3870	5562
PGM-5	1085	3393	4478

注）ステップ数は、コメントを除いた有効ステップの数を示す。

5.1 カプセル化の手順

ソースコードのプロセスをカプセル化する手順を図5-1に示す。

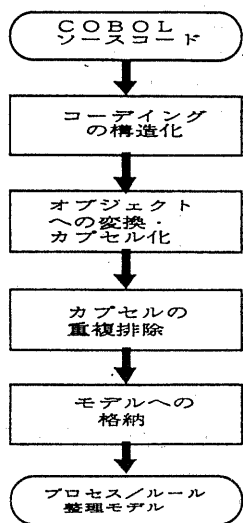


図5-1 カプセル化の手順

(1) コーディングの構造化

構造化されていないソースコードは、GO TO文で処理の流れを変えているため、各データ項目に対する操作や命題判定結果に従う処理の範囲の特定が困難になる。そこで、非構造化ソースコードを構造化して、GO TO文をPERFORM文やELSEに置き換え、上述の問題を解決することにした。カプセル化作業の前処理として、このステップを市販の構造化ツールを用いて実施した。

(2) オブジェクトへの変換・カプセル化

ソースコードをカプセル化する方法は、4章で述べた通りである。これを機械的行なえるよう、ツールを開発し、ソースコードをインプットとし、変換ルールに従い自動的にカプセル化することを可能とした。

(3) カプセルの重複排除

ソースコード内には、同一ステートメントが各所に記述されていることがあるため、分解されたカプセルには重複が生じる。このため、カプセルを構成するオブジェクトおよびメソッドが同一であれば、同じカプセルとして扱い、重複を排除した。

(4) モデルへの格納

以上の処理の後、カプセルをプロセス/ルール整理モデルの5つのテーブルに格納した。

なお、今回開発したツールは、(2)～(4)の処理を手作業を介することなく、一連で実行できるものである。

5. 2 2項オブジェクト表現で扱えないプロセス

実プログラムの中のプロセスをカプセル化していく過程において、上述の2項オブジェクトにて表現できないものがいくつか現れた。その主なものは、

- ・PERFORM
- ・CALL～ USING～
- ・STOP RUN
- ・GOBACK

などである。

PERFORMについては、すでに述べた通り、セクション名やパラグラフ名をカプセル構成要素とし、PERFORMをメソッドとするカプセルで表現した。

CALLについては、USING以下に記述される引き数すべてを1つのカプセルとして集約し、CALL文自体は、サブルーチン名とその引き数カプセルを2項オブジェクトとするカプセルとした。

STOP RUNやGOBACKは、プログラムの制御の終了点（出口点）を示すものである。これらは、メソッドにそれらステートメントを持ち、カプセル構成要素を持たないカプセルとした。

5. 3 IF文以外の制御ステートメント

ソースコード内のプロセスの流れを制御する代表的なステートメントは、IF文であるが、これ以外にも

- ・READ～ AT END～
- ・SEARCH～ WHEN～
- ・PERFORM～ UNTIL～

などが制御をつかさどるステートメントとしてあげられる。これらは、AT END～やWHEN～やUNTIL～が一種の条件判定文であり、この部分だけを見ればIF文と同様の扱いをすべきであるが、READ～やSEARCH～やPERFORM～などの操作に密に関連した制御であることから、これらの記述そのままをメソッドとしてカプセル化することとした。

6 カプセル化の結果

ソースコードのステップ数とそれを変換したカプセルの数との関係を表6-1に示す。ステップ数と重複排除前のカプセル数との関係を見ると、カプセルの数はステップの数の8パーセント増となる。これは、複数条件文が含まれるIF文ではその命題部が1ステップであっても、カプセル化した場合「条件文の数×2-1」に増加することが主な原因である。また、重複排除前後のカプセル数を比較すると、平均で36%減少しており、ソースコード内の約1/3のステートメントは、同一のものが他の箇所にも存在していることがわかった。なお、重複率が高くなる原因の一つとして、同一サブルーチンを複数呼び出すことがあげられる。この場合、サブルーチン呼び出しの前処理・

後処理に同一ステートメントが多数記述されるからである。

表6-1 ステップ数とカプセル数

プログラム名	プロシジャ部の ステップ数	カプセル数			重複率
		重複 排除前	カプセル数 /ステップ数	重複 排除後	
PGM-1	187	199	1.06	138	31
PGM-2	406	433	1.07	382	12
PGM-3	1122	1210	1.08	762	37
PGM-4	1692	1865	1.10	1072	43
PGM-5	1085	1162	1.07	781	33
合計	4492	4869	1.08	3135	36

また、さらに詳しくカプセル化の結果を分析するために、カプセル構成要素の数およびデータ項目からのみなるカプセルの数を調査した（重複排除後）。まず、カプセル構成要素の個数の分布については、表6-2に示す通り2つのオブジェクトから構成されるカプセルが平均70%存在した。これは、ソースコード内のプロセスを2項オブジェクトとして整理したことの結果と考えられる。逆に、2項オブジェクトで整理できないものが残りの30%存在した。1項からなるオブジェクトは、PERFORM文やDISPLAY文が主なものであり、3項以上のものはIF文や、セクションやパラグラフを集約したオブジェクト、あるいはサブルーチンCALLの引数を集約したものなどが代表的なものである。

表6-2 カプセルの構成要素数の分布（%）

プログラム名	構成要素数			
	1	2	3	4
PGM-1	11	62	15	12
PGM-2	8	78	8	6
PGM-3	7	70	8	15
PGM-4	7	70	8	15
PGM-5	8	70	8	14
平均	7	70	11	12

また、カプセル構成要素の種類別個数の分布を示したのが、表6-3である。データ項目からのみなるカプセルは、平均64%存在する。これらのほとんどは、2項オブジェクトとしてカプセル化されたものである。また、カプセルのみを構成要素とするカプセルは、平均33%あり、これらは前述の制御オブジェクトとしての性格を持つもの（IF文など）と単にプロセスを集約するためのもの（セクションやパラグラフ）とからなる。これから、ソースコード内のステートメントの約1/3は、所定の結果を得るために必要な制御に費やされていることがわかった。なお、データ項目からのみなるカプセルの中でも、フラグやカウンタなど制御のためのデータ項目をその構成要素としているカプセルも多数ある。例えば、PGM-3では、全IF文の90%はプロセスを制御するためのもので、残りの1

0%が当該アプリケーションでの業務ルールとなるものであった。このことを考慮すると、所定の結果を得るために必要な制御に費やされているステートメントは上述の1/3よりもさらに増加する。

表6-3 カプセル構成要素種別の分布（%）

プログラム名	データ項目	カプセル	その他
PGM-1	57	40	3
PGM-2	70	26	4
PGM-3	64	33	3
PGM-4	61	37	2
PGM-5	66	32	2
平均	64	33	3

上記に述べたカプセルは、他カプセルとの関係、すなわちカプセル同志の包含関係をも表現しているため、その関係を表現するために図6-1に示すカプセル関連図を用いた。この表現形式を用いることによって、各カプセルとそれに対してトリガーとなるIF文との関係がたどれることになる。すなわち、あるカプセルが実行されるためのトリガーがどうかといった場合、そのカプセルから上流方向にIF文を探索していくことにより、所定のトリガーすべてが見いだされる。

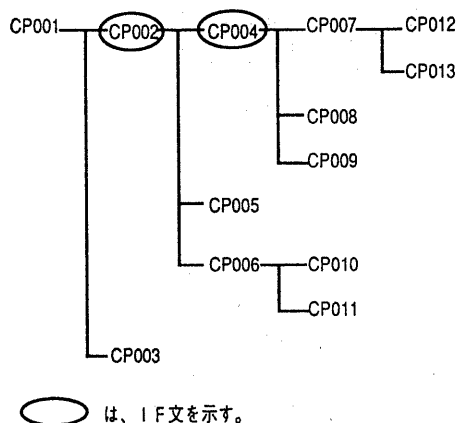


図6-1 カプセル関連図

7 業務ルールの抽出

以上のカプセル化手法を用いて、ソースコードから実世界での業務処理に関するルール（業務ルール）の抽出を試行した例を示す。

ソースコード内に、図7-1に示すようなロジックがあった場合、定価の30%引きで販売するのは、どのような顧客であるかという業務ルールを抽出することを例に説

明する。このロジックからは、正会員で、購入額が30万円以上で、分割払いでない場合に、30%引きの販売となる。このルール抽出は、現在着目する処理（売値＝定価＊0.7）を起点に条件判断箇所を上流にたどり、その該当する条件判断箇所のみを選択していけば（図7-2）、その処理の業務ルールが明らかになると考えた。

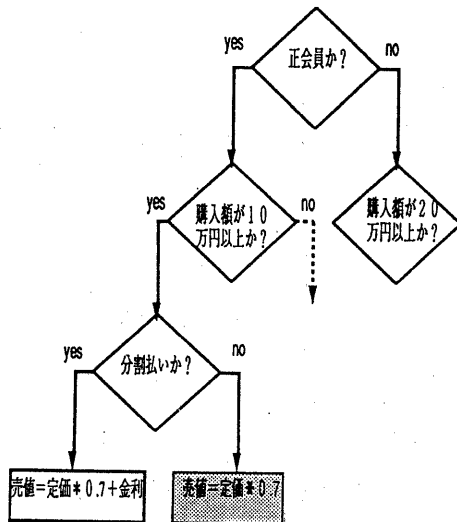


図7-1 売値算出のロジック

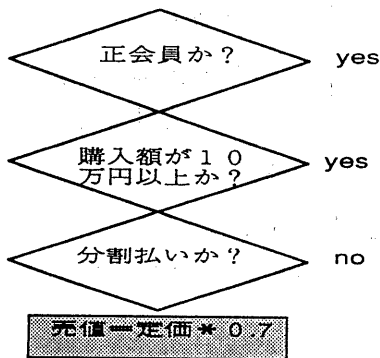
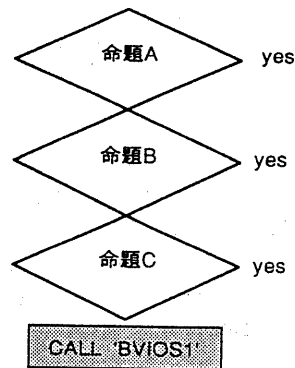


図7-2 30%引きの売値とする業務ルール

前章までに述べた、ソースコードを個別のカプセルに変換することにより、ここで述べた業務ルールの抽出が実現しやすくなっている。実ソースコードで実際にルールの

抽出を行なった結果を図7-3に示す。



命題A: C-DB-FOUND

命題B: ND-UPDATE-CTR-I1=K-COUNTER-S-W1

命題C: ND-SHIJI-I1=000SIJI-TEISEI

↓ データ分析結果から

命題A：キーがDBに存在するか？

命題B：現在の更新回数が、前画面表示時の更新回数と等しいか？

命題C：画面からの指示が「訂正」か？

図7-3 ルール抽出の実例

ここでは、DBへのアクセスを行なうサブルーチンが起動されるためのルールについて、その抽出を試みた。具体的には、まずサブルーチン呼び出しステートメントを起点に、カプセル関連図を上流側にたどって行く。そこで現われたIF文のカプセルを抜き出し、図7-3の通り図化することで、ルールが抽出された。このルールの持つ意味については、データ分析結果から求めた各データ項目の持つ意味を用いて解釈を試みた。その結果、

- ・指定したキーがDBに存在し、かつ
- ・そのレコードの現在の更新回数が、前画面表示時の更新回数と等しく、かつ
- ・画面からの指示が「訂正」である

時、サブルーチンが起動されると意味解釈ができた。

このルールは、業務ルールではないが、この抽出作業を行ない、各ルールの意味を解釈することで、業務ルールのみを選別することが可能と考える。

なお、ここで述べた方法はルールの抽出について必ずしも網羅性が十分であるとの確認までには至っていない。また、ルールの意味解釈を可能とするためには、各データ項目の持つ意味を事前に明らかにしておくことが非常に重

要であるが、その作業は一般的にかなりの業務知識と労力を要するものである。しかしながら、上述の仮説は、データ中心分析に基づきソースコードからルールの抽出を行ない、意味解釈する一つの指針となるものであり、今後さらに検討を加え、課題を克服したいと考える。

8 おわりに

データ中心分析に基づき、ソースコードをオブジェクトに変換する手法について述べた。ここでは、2項オブジェクトと呼ぶ形態でカプセル化することの可能性および妥当性が確認できた。そして、それらカプセルを格納するプロセス/ルール整理モデルの構造を定義した。

また、カプセル化することにより、今回試行に用いた既存プログラムでは、約1/3が同一ステートメントで重複していたこと、70%程度が2項オブジェクトにて整理できること、さらには全ステートメントの1/3以上は、所定の結果を得るために必要な制御に費やされていることなどがわかった。

業務ルールの抽出については、ソースコードの変換形であるカプセルからそこに埋め込まれたルールを抽出し、その意味を解釈する方法の試行結果を示した。この方法については、条件判断文を累積させることによって、業務ルールの抽出および意味解釈が可能となるという仮説を設定した。この仮説に関しては、複雑なロジックの場合やルールの発見の精度を高めるなど、今後の課題は数多く残っている。特に課題となるのは、各データの持つ意味をいかに容易に知り得るかということである。これら課題を解決し、あるいは、できるだけ人の判断を要する作業を小さくしつつ、リバースエンジニアリングの方法論および分析手順の確立を実現したいと考える。

参 考 文 献

- 1) 堀内 一 : データ中心システム設計、オーム社 (1988)