U.D.C. 681.32.068:800.92 COBOL

データに着目した仕様を入力とするCOBOLプログラム生成システム“DSL”の開発

Development of the Application Program Generator Based on the Data Oriented Specification Language “DSL”

河野　史男* *Fumio Kôno*
鵜飼　純一* *Jun'ichi Ugai*
中尾田　操** *Misao Nakaoda*
岸　　一也** *Kazuya Kishi*

従来のソフトウェア開発方式では，コンピュータ知識をベースとした要求の記述，要求からプログラムへの人手による変換，プログラムとドキュメントの一致化の難しさなどにより，ソフトウェアの開発効率，品質を飛躍的に高めることは難しい。このため，利用者自身による業務知識を前提とした要求の記述，記述結果から自動的にプログラムを生産する一貫した方式が望まれている。今回この目的に沿って，ビジネスアプリケーションを対象に，従来の手続形ではなく入出力のデータを中心に仕様を定義する方式，及びこれを入力としてCOBOLプログラムを自動生成するシステムを開発した。具体アプリケーションの開発に本ツールを適用した結果，システム設計からテスト工程までに要する工数を従来の約$\frac{1}{2}$に短縮できる見通しを得た。

1 緒　　言

ソフトウェアの生産性向上の必要性が認識されてから久しくなる。この間，ソフトウェア開発の各段階で，開発作業を支援する各種の方法論，技法，ツールの研究開発及び実用化が盛んに行なわれてきた。

この結果，ビジネス分野を対象としたアプリケーションソフトの開発では，プログラミングフェーズを中心に，高生産性言語，プリコンパイラなどが数多く実用化されており[1]，またこれらを取り入れ，デバッグテスト，保守フェーズを含んだ一貫した開発環境も整えられてきている。

一方，上流工程のニーズ分析，計画，設計フェーズについては，汎用的な方法論，要求仕様の定義方式に関するものが提案されてはいるが，上記プログラミングツールとの間にはギャップがあり，一貫した開発環境が出来上がっていないのが現状である。

そこでソフトウェアの要求仕様定義技法とプログラムの生産技術とを融合したアプリケーションソフトの生産方式として，データ中心形の仕様定義技法“DSL”(Data Oriented Specification Language)とこれに基づくプログラムゼネレータを試作開発した。今回開発したゼネレータはビジネスアプリケーションのバッチ処理分野を適用対象としたものである。このシステムの概要を**図1**に示す。

2 DSLの仕様定義方式

2.1 DSLの仕様定義の考え方

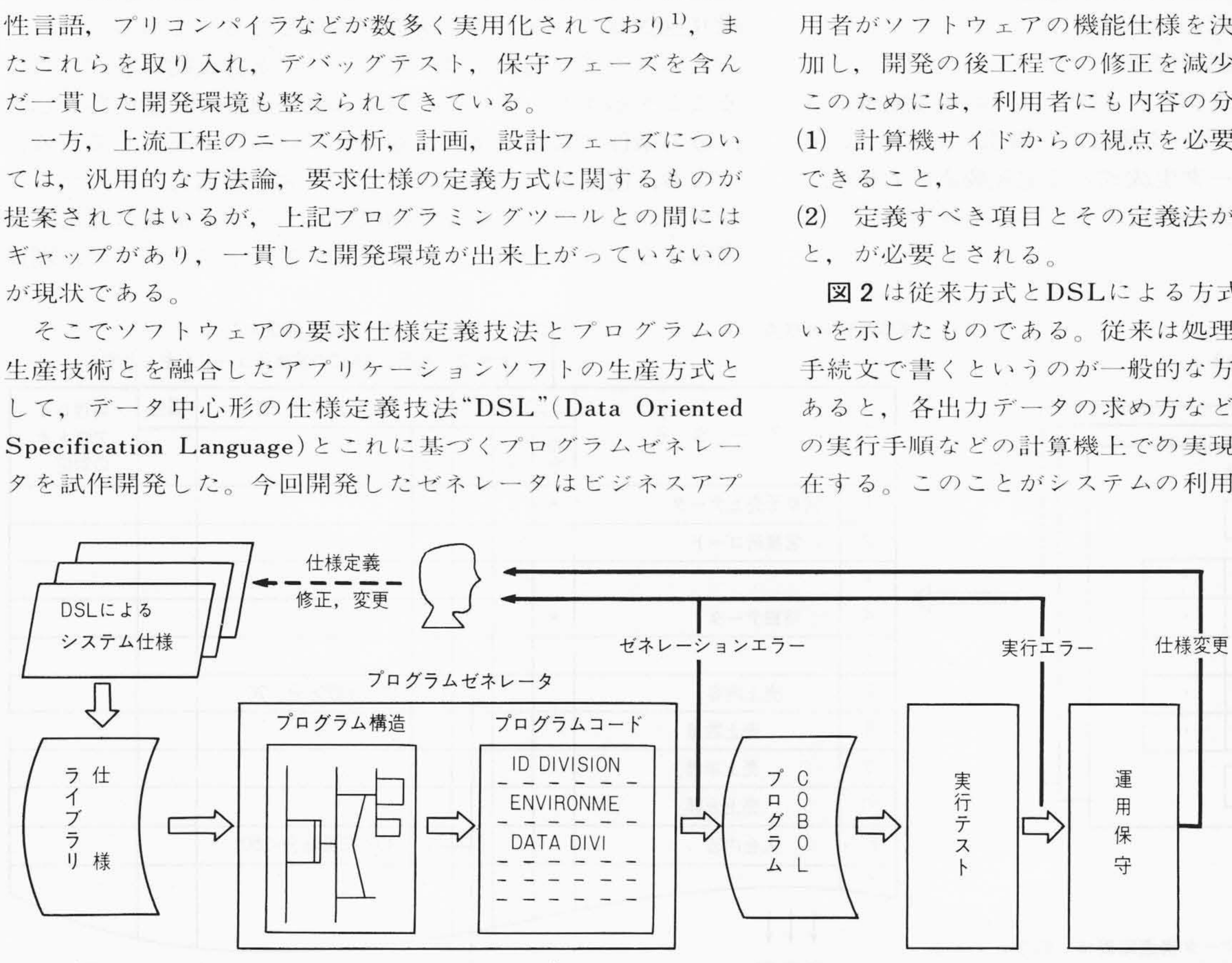
ソフトウェアの生産性向上を図るためには，システムの利用者がソフトウェアの機能仕様を決める段階から積極的に参加し，開発の後工程での修正を減少させることが必要である。このためには，利用者にも内容の分かる定義法，すなわち，

(1) 計算機サイドからの視点を必要とせず，業務知識で定義できること，

(2) 定義すべき項目とその定義法が，明確かつ容易であること，

が必要とされる。

図2は従来方式とDSLによる方式について，定義内容の違いを示したものである。従来は処理プロセスを中心に仕様を手続文で書くというのが一般的な方法であった。この方式であると，各出力データの求め方などの本来の業務内容と処理の実行手順などの計算機上での実現方法に関する内容とが混在する。このことがシステムの利用者にとって，仕様を定義



注：略語説明　DSL(Data Oriented Specification Language)

図1　プログラムゼネレータの概要　このゼネレータは，定義されたデータ中心のシステム仕様を入力することによって，プログラム構造，更にCOBOLのソースプログラムを自動生成する。仕様はすべて仕様ライブラリに蓄積しておき，テスト，運用保守はすべてこの仕様を対象として行なう。

* 日立製作所システム開発研究所　** 日立製作所ソフトウェア工場

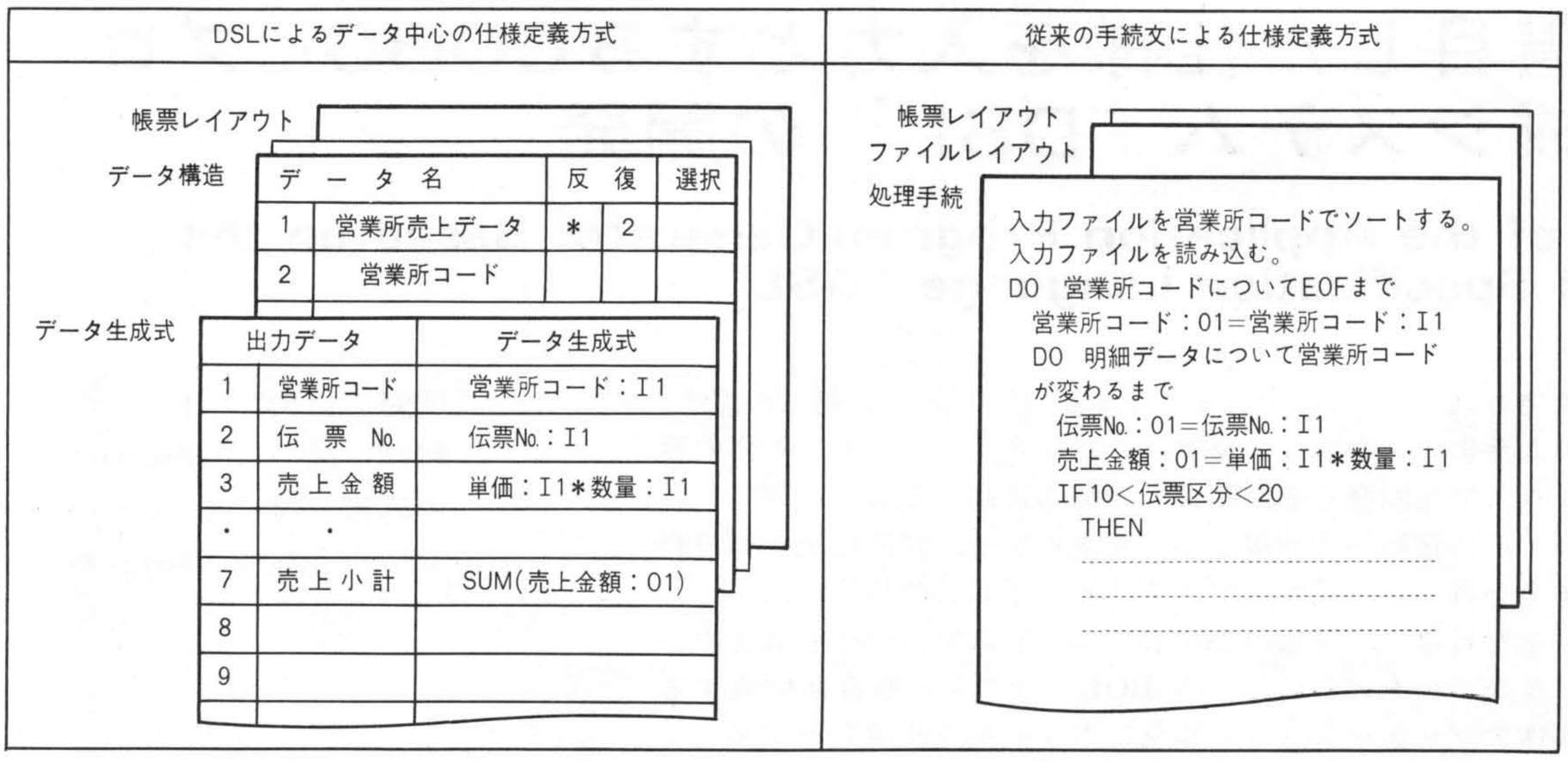


図2 DSLと従来方式の仕様定義の違い
従来方式が処理手続中心に仕様を定義するのに対し，DSLは，データ中心に仕様を定義する。これにより定義者は処理の順序などを考えることなく，データ項目対応に業務上必要な事項を定義することができる。

したり理解することを難しくしている。

これに対しDSLでは，業務知識に基づいた定義方式[2)~4)]を用いている。定義の主な内容は，

(1) データ生成式：出力データの求め方を実行順序を考慮することなく，データ項目ごとに独立して定義するもの。

(2) データ構造：入出力のデータ項目及び各データの論理的な配列関係を定義するもの。

である。DSLではこれらの定義内容から，ゼネレータにより計算機上で実行可能なプログラム手続へと変換する。

上記の考え方を基に，定義の容易化，正確化及び計算機入力を可能とするため，

(1) 定義内容を明確に規定した定義用ワークシート

(2) データ生成式などの記述文法

を定めている。DSLによるシステム仕様は，(a) 環境定義，(b) デシジョンテーブル，(c) データ構造，(d) 帳票レイアウト，(e) データチェック式，(f) データ生成式の各定義要素から構成している。以下，各定義要素についてその内容を述べる。

2.2 データ構造定義

データ構造は，データの階層，反復，選択の三つの要素を用いることによって，入出力データについての論理的な配列関係を表現しようとするものである。

図3は，売上日計表の伝票の例と，これについてのデータ構造の定義例を示したものである。同図でデータの階層関係とは，第1レベルに営業所データが，第2レベルに営業所コードや明細データが位置づけられるという構造を示し，これを定義シート上ではデータの字下げ(Indentation)で表現している。データの反復とは，営業所データが営業所コードを単位として繰り返され，更にある営業所の中は，明細データが繰り返されているという構造を表現したものである。

次にデータの選択とは，複数データが背反的に存在していることを示すもので，この例では売上内容と入金内容が伝票区分の条件により背反的に存在していることを示している。

なお，同定義シート上には，データの媒体名とか，データの属性，レイアウトなどの物理情報をデータ構造に対応させ定義することとしている。帳票レイアウトについては，別途

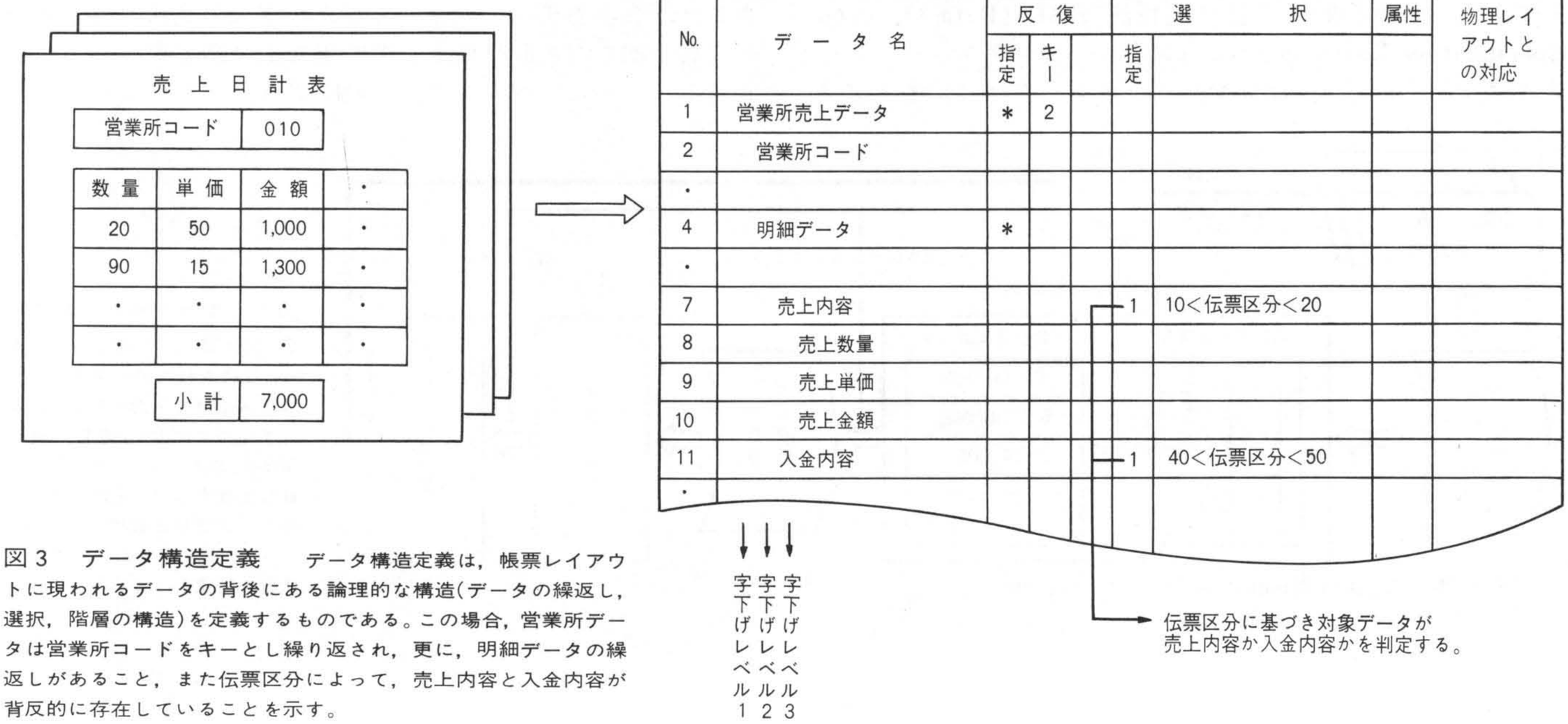


図3 データ構造定義 データ構造定義は，帳票レイアウトに現われるデータの背後にある論理的な構造(データの繰返し，選択，階層の構造)を定義するものである。この場合，営業所データは営業所コードをキーとし繰り返され，更に，明細データの繰返しがあること，また伝票区分によって，売上内容と入金内容が背反的に存在していることを示す。

帳票レイアウト定義シートに定義し，データ構造定義シート上に対応づけの情報を記述することとしている。レイアウト定義の例は，**5**章適用事例の**図9**に示すとおりである。

2.3　データの生成式及びチェック条件の定義

出力データの各項目について，その生成の由来をデータ生成シートに，また入力データについては入力資格条件をデータチェックシート上に定義する。この定義方法は，従来の手続文で記述するのではなく，データ中心に記述する。このための定義文法の例を以下に示す。

(1)　データ名称

データ名称を生成式やチェック条件式の中でユニークにするため，次のような形式でデータ名称にファイルの区分コード(I：入力，O：出力，F：参照)を付加し使用する。

売上数量：F 1 ……(1)

(2)　データ生成式

データの生成式は四則演算子，組込関数，条件式を用いて出力項目単位に定義することができる。なお，従来の手続文とは異なり，各生成式の順序づけは不要である。次式は演算の例である。ここで左項は出力データ項目を示している。

売上金額｜売上数量：I1＊売上単価：F2……(2)

DSLはビジネス分野でよく用いられる合計計算をはじめとする，各種の決まった手続を組込関数として設けている。次式は合計計算の例であり，商品コード別に売上金額を合計するという生成式を示している。

売上小計｜SUM(売上金額：O1 // 商品コード：O1)…(3)

同一出力項目に対して複数の生成式が存在する場合は，次のように生成式の前に条件式を定義する。

金額｜(区分：I1＝0)→売上金額：I1,
　　　条件式　　生成式
　　OTHER　　→−1＊売上金額：I1……(4)

外部ファイルのデータの参照条件も同形式で表現する。

商品名称｜(商品コード：F1＝商品コード：I1)→
　　商品名称：F1……(5)

以上のように非手続的な形式で通常の業務は定義可能であるが，どうしても手続的に仕様を定義せざるを得ない場合は，別紙という関数を設け，COBOL言語によるオウンコーディングの受け口を設けている。なお生成式定義の例は，**5**章適用事例の**図10**に示している。

(3)　データチェック条件

これも生成式の文法に準じ，入力データの資格条件を定義することができる。次式は入金コードが10，15，20のいずれかでなければならないという条件を示した例である。

入金コード｜＝10　or　15　or　20……(6)

2.4　その他の定義要素

(1)　環境定義：対象プログラムに対する入出力環境(入出力ファイル名など)を定義するものである。なお，環境定義の例は，**5**章適用事例の**図7**に示すとおりである。

(2)　デシジョンテーブル定義：入力データなどの条件によって生成式定義シートが複数個に分類できる場合，この分類条件をデシジョンテーブルを用いて定義できる。**図4**はファイル更新の場合のデシジョンテーブルの定義例と，他の定義シートとの関係を示したものである。

3　プログラムゼネレーションの考え方

上記のデータ中心の仕様からプログラムコードを生成する基本的な考え方として，入出力のデータ構造からプログラム構造を導き出すというジャクソン法の考え方[5]を用いている。

データ構造からプログラム構造を生成する概念図を**図5**に示す。この方法の基本は，出力データ構造上の反復，字下げ，選択の各構造を，プログラム上の繰返し，階層，判定の各構造に対応させてプログラム構造を生成するという考え方である。次に，この構造上の出力データに対して該当するデータ生成式を組み込むことによって初期のプログラム構造を生成する。

更にこのプログラム構造を完成させるために，次の手順を踏む[6]。

(1)　入力データ構造を出力データ構造に変換するための，中間ファイルの導入及び入出力タイミングの決定

(2)　データ生成式のプログラム手続文変換

以上のような方法でプログラム構造が出来上がると，最終的にこの内容を基にCOBOLのプログラムコードを編集していきプログラムを完成させる。

4　DSLの利用手順

4.1　仕様定義上の主な制約

DSLは対象プログラムについての入出力のデータ構造を定義することによって，生成式上で手続的な記述を用いないで入出力データを直接的に結びつけることを可能としている。

しかし，入力と出力のデータ構造間の差が大きいと入力から出力データへの構造変換が何度も発生し，途中で中間的なデータ構造を介在させない限り入出力データを直接的に関係づけるのは難しい。したがって，DSLでは一度に定義する仕様の単位として，入出力データが直接的に対応できる範囲，すなわち出力から見て最初にデータ構造の変化が起こるとこ

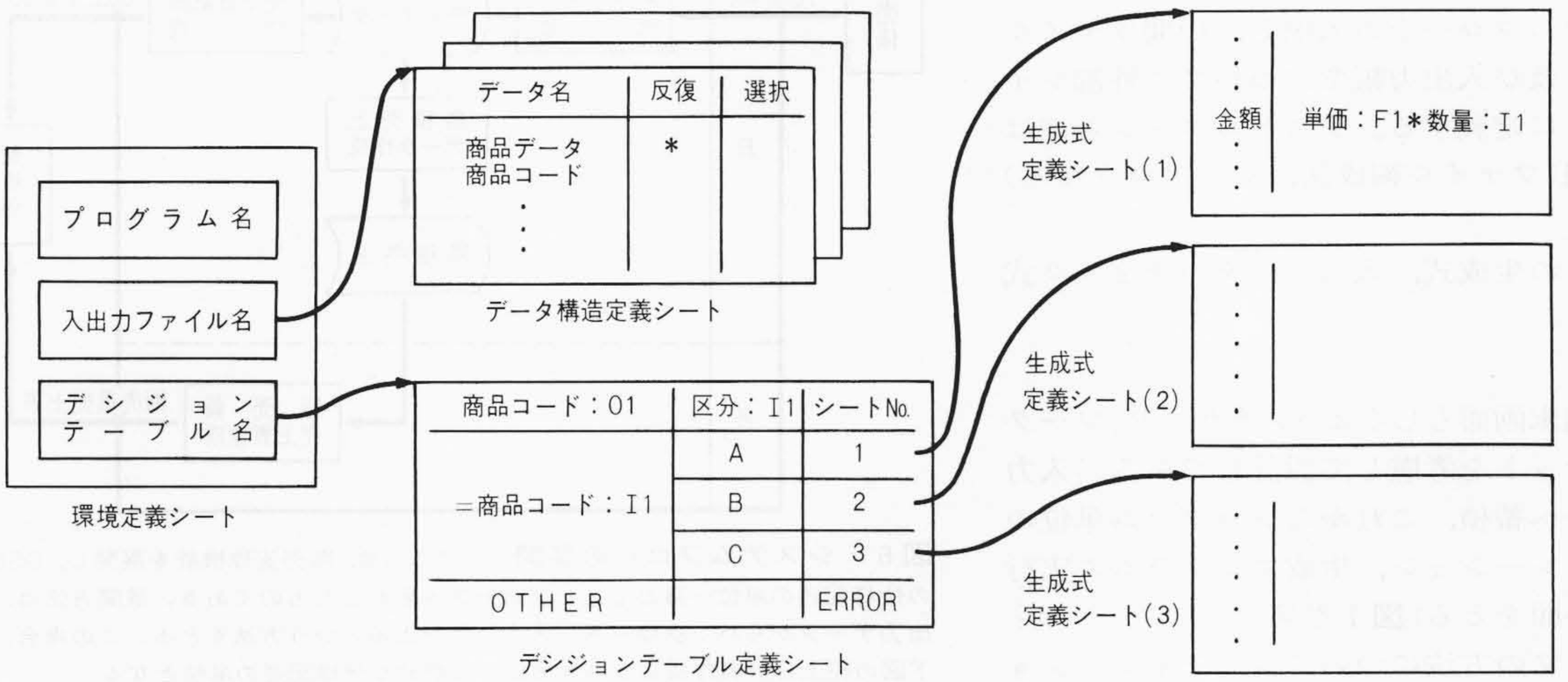


図4　デシジョンテーブル定義シートの位置づけ
デシジョンテーブルは，入力データなどの条件によって，生成式シートが複数に分類できる場合，その分類条件を定義するものである。この図の場合，入力データである商品コード，区分の値によって生成式シートを三つに分類する条件を定義している。

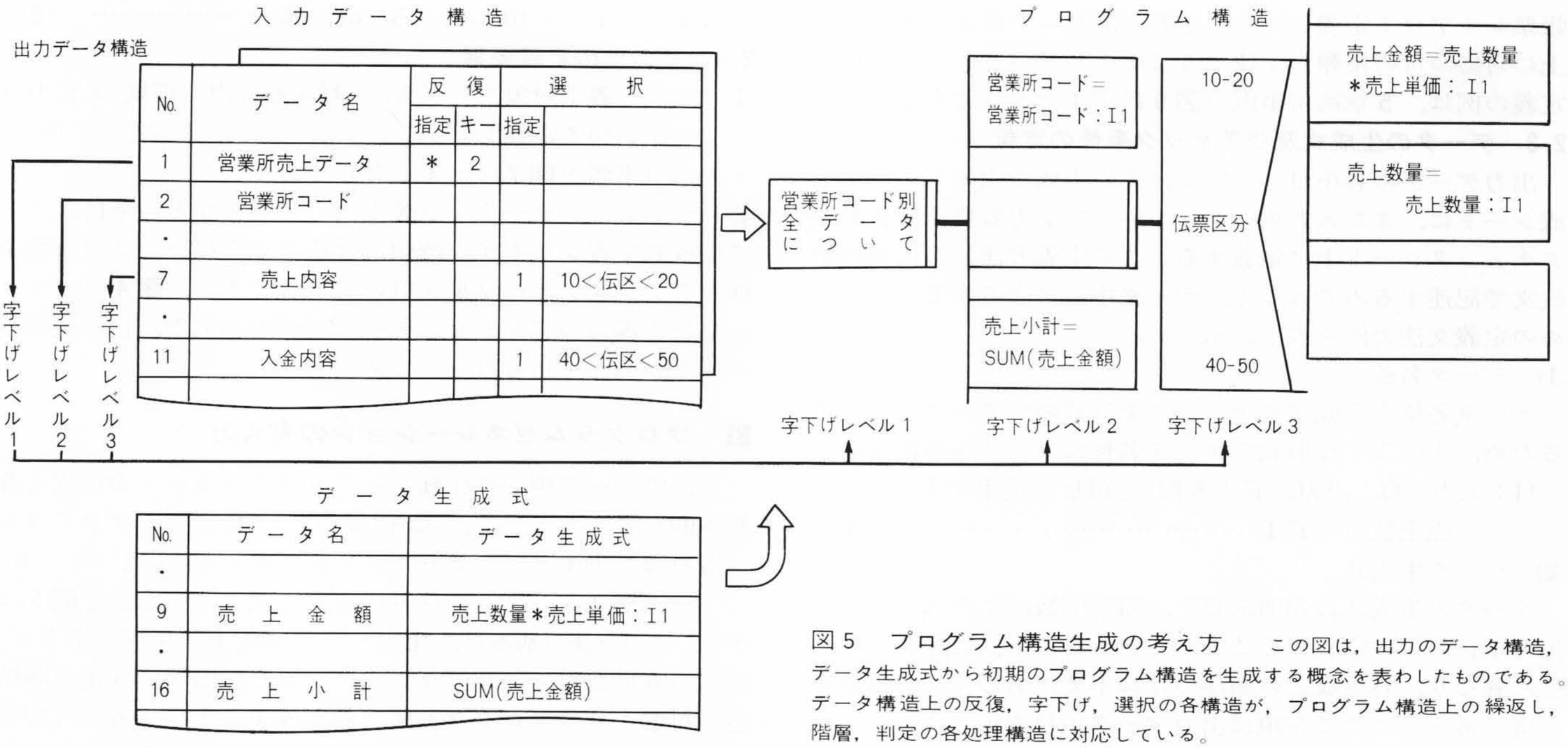


図5 プログラム構造生成の考え方 この図は，出力のデータ構造，データ生成式から初期のプログラム構造を生成する概念を表わしたものである。データ構造上の反復，字下げ，選択の各構造が，プログラム構造上の繰返し，階層，判定の各処理構造に対応している。

ろで入力データを設定することとする。こうして次節で述べるように，出力から順次中間データ構造を取り決め，初期入力にたどりつくというシステムフローの展開方式をとる。

仕様定義上のもう一つの制約は，DSLが出力中心の仕様の展開方式を用いていることから，定義対象の出力データ構造は一つとするというものである。

4.2 仕様定義手順

仕様定義はエンドユーザーが中心になって進め，システムエンジニアはシステムサイドから定義結果の適正化や物理情報の追加定義を行なう[7),8)]。以下，定義手順の概要をエンドユーザーの定義を中心に示す。

(1) システムフローの作成(**図6**参照)

(a) システム化対象業務機能，及びその入出力情報を明確にする。

(b) 該当業務機能について，出力データから入力データへとバックワードに仕様定義の単位(プログラムの単位)を明らかにしシステムフローを作成する。このとき，1出力など前述した定義上の制約を考慮して中間のデータ構造(ファイル)を設けていく。

(c) システムエンジニアはこの結果に対し，全体の運用効率を考慮して必要とあればソート処理の挿入箇所の修正などフローを適正化する。

(2) DSLによる仕様記述

(a) 展開されたシステムフロー上の入出力，中間ファイルについてデータ構造，及び入出力帳票についての外部レイアウトをワークシートに定義する。システムエンジニアはこれに対して物理情報(ファイル編成法，レイアウトなど)を追加定義する。

(b) 同じく出力データの生成式，入力データのチェック式を定義する。

4.3 ゼネレータの運用

定義された仕様は，端末画面もしくはパンチカード(ワークシートはカードフォーマットを考慮して設計してある。)入力により仕様ライブラリーへ蓄積，これからプログラム単位の仕様一式を取り出しゼネレーション，生成プログラムの実行テスト，運用・保守の手順をとる(**図1**参照)。

このうち仕様のデバッグの方法については，ゼネレーション結果のエラーメッセージを参照することによって，該当箇所を修正，再入力，再ゼネレーションというプロセスをとる。プログラムを実行した結果のエラーについても，プログラムの手続を追うのではなく基本的に仕様と実行結果とを照合することによってデバッグする。

以上のようにゼネレータを用いることによって，プログラム設計，製造，デバッグの工程をたどることなく，システム仕様の入力とこれのデバッグによってプログラムの開発を可能にする。開発後のプログラムの保守の問題についても同じ手順で対応できる。

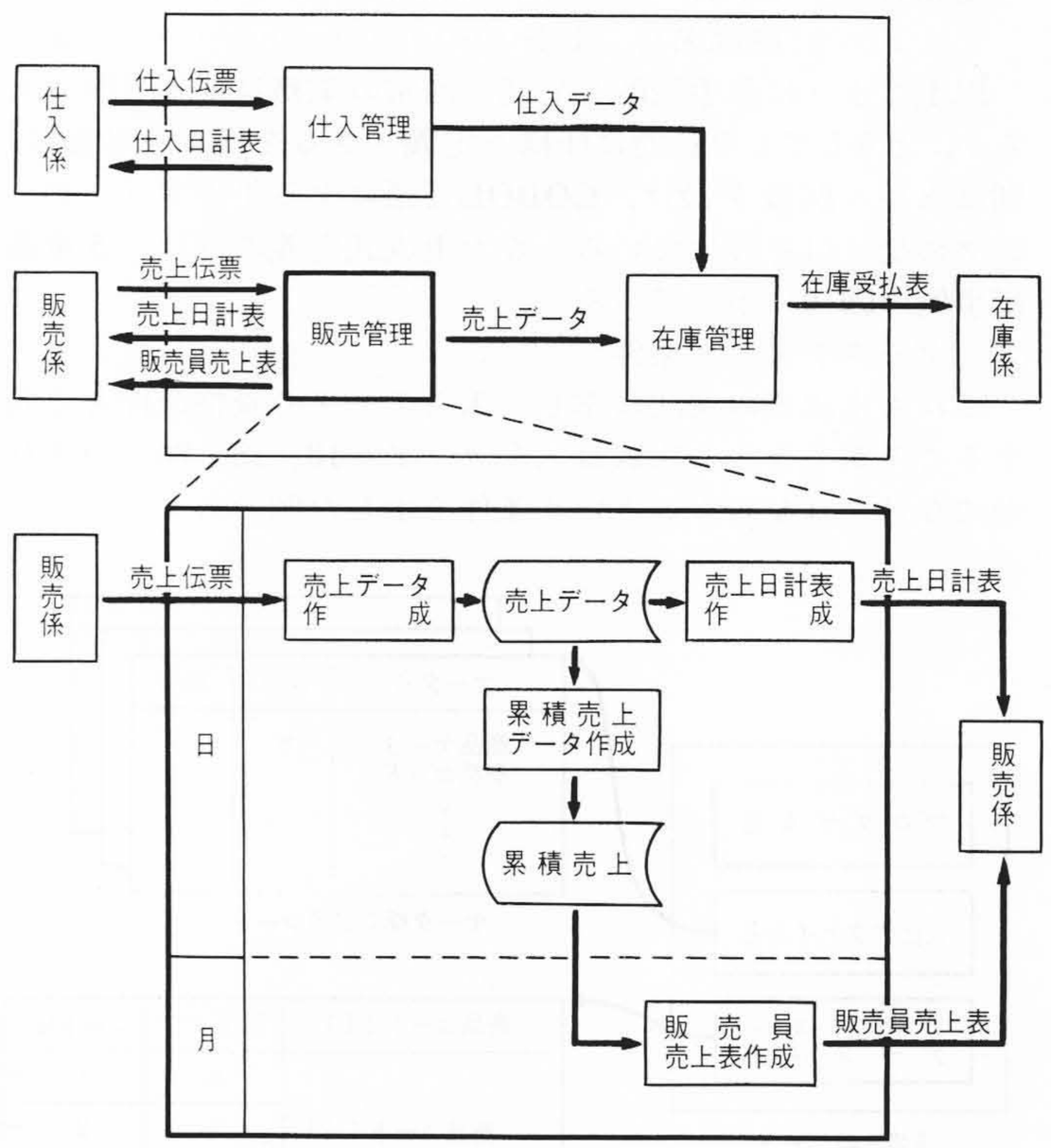


図6 システムフローの展開 この図は，販売管理機能を展開し，DSLの仕様記述の単位へ落とし込むプロセスを表わしたものである。展開方法は，出力データからバックワードに入力にさか上るという方法をとる。この場合，下図の売上日計表作成などの各機能が最終的な仕様記述の単位となる。

5　適用事例

DSLの適用事例として，売上日計表作成のプログラムについてのDSLの仕様定義例とゼネレーション結果のCOBOLプログラムの一部を**図7～11**に示す。このプログラムは，売上データを入力し，商品マスタ，得意先マスタを参照して，売上日計表を出力するものである。

ここで**図7～10**は当プログラムに対する仕様を定義したワークシートの内容を示し，**図11**はこれらを入力することによって自動生成されたCOBOLプログラムを示している。この内容は以下のとおりである。

(1)　環境定義(**図7**参照)

処理区分：　ｻｸﾋｮｳ　｜　プログラム名：　ｳﾘｱｹﾞﾆｯｹｲﾋｮｳｻｸｾｲ　｜　URINIK　｜　設　計：　ｳｶｲ　ｼﾞｭﾝｲﾁ830401　｜　審　査：　｜　承　認：

	参照ファイル	媒体
F1	ﾄｸｲｻｷｰﾏｽﾀｰ	D
F2	ｼｮｳﾋﾝｰﾏｽﾀｰ	D

	入力ファイル	媒体
I1	ｳﾘｱｹﾞﾃﾞｰﾀ	D

	出力ファイル	媒体
O1	ｳﾘｱｹﾞﾆｯｹｲﾋｮｳ	L

図7　環境定義の例　売上日計表作成プログラムについての入出力の環境を定義したものである。ここで入出力ファイルの媒体の記号はD：ディスク，M：テープ，L：リスト，C：カードで表わしている。

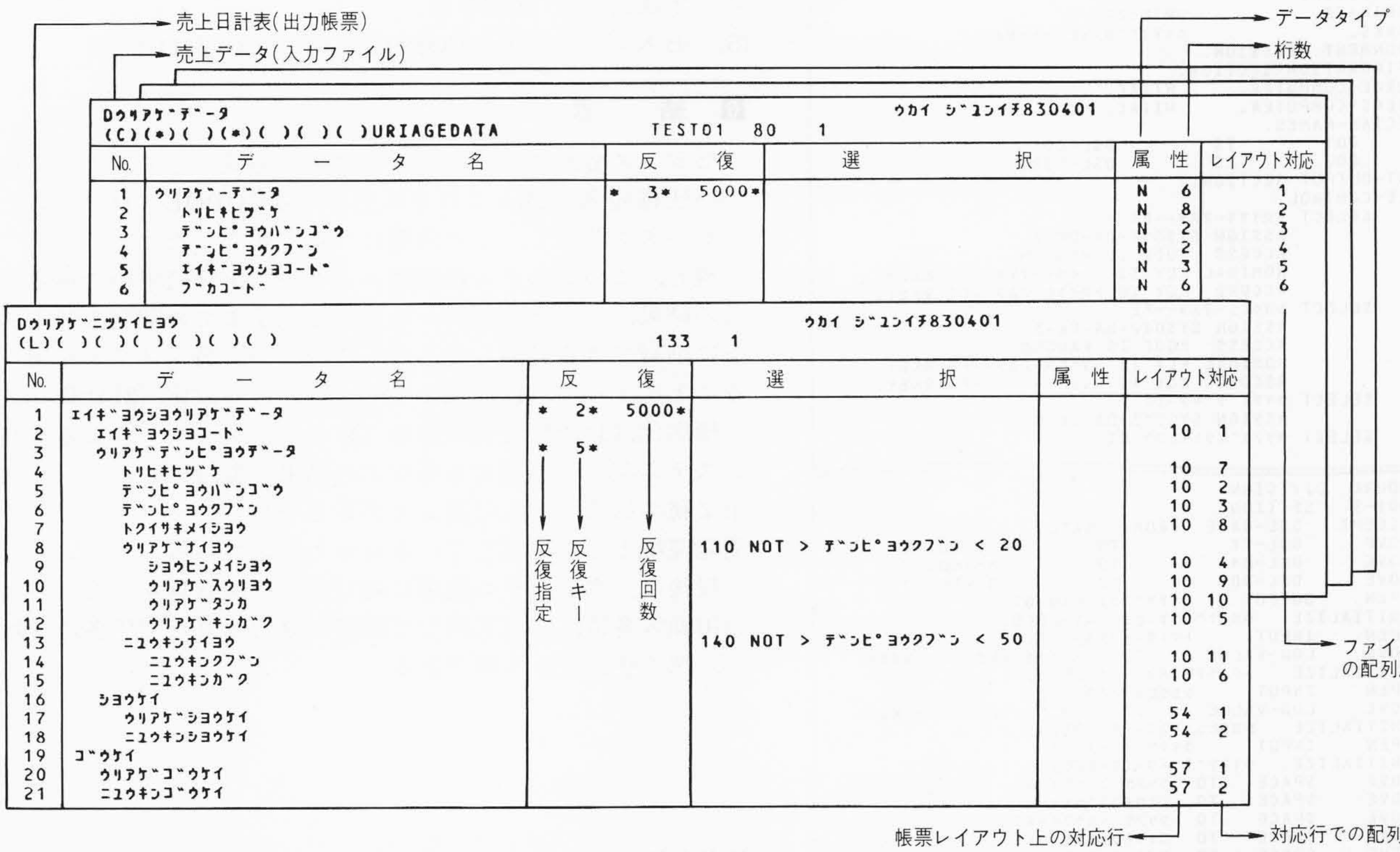


図8　データ構造の定義例　当事例で使用する入出力，参照の各ファイルについて，それぞれデータ構造を定義する。売上日計表，売上データで属性，レイアウト対応の内容が異なるのは，対象の媒体が帳票とファイルの違いによる。帳票の場合，属性の欄は記入せず帳票レイアウト上に編集形式として記述する(図9参照)。

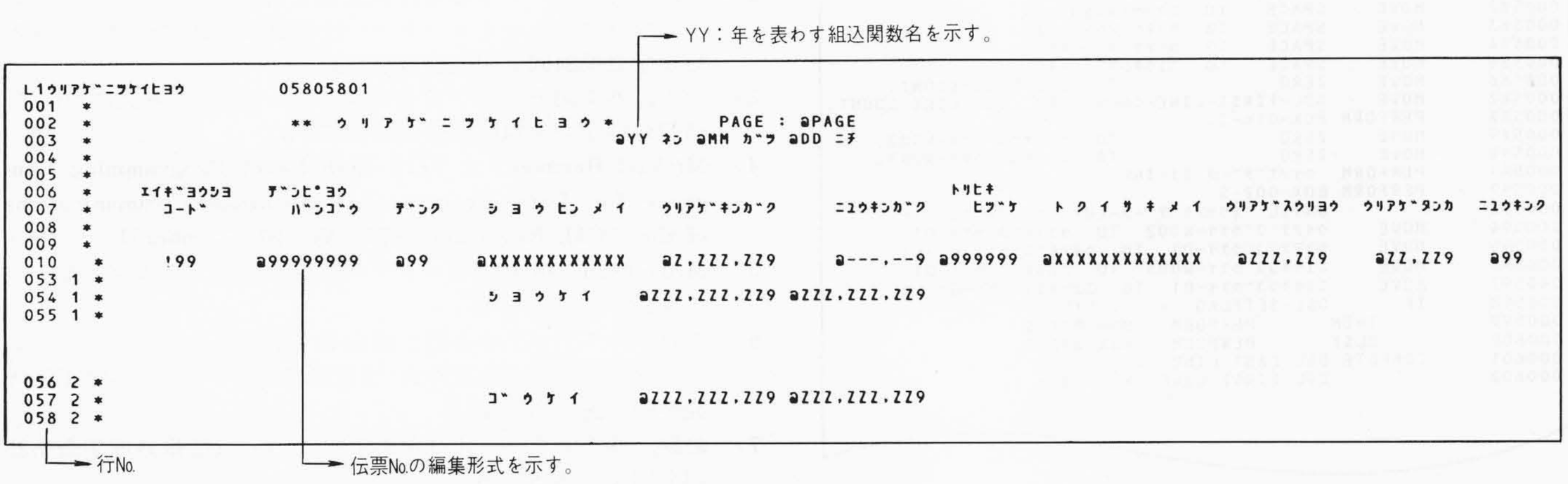


図9　帳票レイアウト定義の例　出力である売上日計表のレイアウトを定義したものである。内容としては，各データのレイアウト，編集形式，見出しなどである。データ構造上のデータとの対応づけは，データ構造定義シート上で行なう。

No.	出力データ	データ生成式
1	ｴｲｷﾞｮｳｼｮｳﾘｱｹﾞﾃﾞｰﾀ	
2	ｴｲｷﾞｮｳｼｮｺｰﾄﾞ	
3	ｳﾘｱｹﾞﾃﾝﾋﾟｮｳﾃﾞｰﾀ	
4	ﾄﾘﾋｷﾋﾂﾞｹ	
5	ﾃﾝﾋﾟｮｳﾊﾝｺﾞｳ	
6	ﾃﾝﾋﾟｮｳｸﾌﾞﾝ	
7	ﾄｸｲｻｷﾒｲｼｮｳ	(ﾄｸｲｻｷｺｰﾄﾞ:F1 = ﾄｸｲｻｷｺｰﾄﾞ:I1) -> ﾄｸｲｻｷﾒｲｼｮｳ:F1
8	ｳﾘｱｹﾞﾅｲﾖｳ	
9	ｼｮｳﾋﾝﾒｲｼｮｳ	(ｼｮｳﾋﾝｺｰﾄﾞ:F2 = ｼｮｳﾋﾝｺｰﾄﾞ:I1) -> ｼｮｳﾋﾝﾒｲｼｮｳ:F2
10	ｳﾘｱｹﾞｽｳﾘｮｳ	
11	ｳﾘｱｹﾞﾀﾝｶ	
12	ｳﾘｱｹﾞｷﾝｶﾞｸ	ｳﾘｱｹﾞｽｳﾘｮｳ:01 * ｳﾘｱｹﾞﾀﾝｶ:01
13	ﾆｭｳｷﾝﾅｲﾖｳ	
14	ﾆｭｳｷﾝｸﾌﾞﾝ	
15	ﾆｭｳｷﾝｶﾞｸ	(ﾃﾝﾋﾟｮｳｸﾌﾞﾝ:I1 = 41) -> ﾆｭｳｷﾝｶﾞｸ:I1 * -1 , OTHER -> ﾆｭｳｷﾝｶﾞｸ:I1
16	ｼｮｳｹｲ	
17	ｳﾘｱｹﾞｼｮｳｹｲ	SUM(ｳﾘｱｹﾞｷﾝｶﾞｸ:01//ｴｲｷﾞｮｳｼｮｺｰﾄﾞ:01)
18	ﾆｭｳｷﾝｼｮｳｹｲ	SUM(ﾆｭｳｷﾝｶﾞｸ:01//ｴｲｷﾞｮｳｼｮｺｰﾄﾞ:01)
19	ｺﾞｳｹｲ	
20	ｳﾘｱｹﾞｺﾞｳｹｲ	SUM(ｳﾘｱｹﾞｷﾝｶﾞｸ:01//ALL)
21	ﾆｭｳｷﾝｺﾞｳｹｲ	SUM(ﾆｭｳｷﾝｶﾞｸ:01//ALL)

図10 生成式定義例 各出力データについて，項目対応にその生成の由来を定義したものである。

```
000001 CBL QUOTE,STATE,FLOW
000002 ID  DIVISION.
000003  PROGRAM-ID.            URINIK2.
000004  REMARKS.               ｳﾘｱｹﾞﾆｯｹｲﾋｮｳｻｸｾｲ.
000005 ENVIRONMENT DIVISION.
000006  CONFIGURATION SECTION.
000007   SOURCE-COMPUTER.      HITAC.
000008   OBJECT-COMPUTER.      HITAC.
000009   SPECIAL-NAMES.
000010         C01       IS        DSL-C01
000011         C02       IS        DSL-C02.
000012  INPUT-OUTPUT SECTION.
000013   FILE-CONTROL.
000014        SELECT ﾄｸｲｻｷｰﾏｽﾀｰｰF1
000015               ASSIGN SYS011-DA-DK-I
000016               ACCESS  MODE IS RANDOM
000017               NOMINAL KEY IS ﾄｸｲｻｷｰﾏｽﾀｰｰF1-NKEY
000018               RECORD  KEY IS ﾄｸｲｻｷｰﾏｽﾀｰｰF1-RKEY.
000019        SELECT ｼｮｳﾋﾝｰﾏｽﾀｰｰF2
000020               ASSIGN SYS012-DA-DK-I
000021               ACCESS  MODE IS RANDOM
000022               NOMINAL KEY IS ｼｮｳﾋﾝｰﾏｽﾀｰｰF2-NKEY
000023               RECORD  KEY IS ｼｮｳﾋﾝｰﾏｽﾀｰｰF2-RKEY.
000024        SELECT ｳﾘｱｹﾞﾃﾞｰﾀｰI1
000025               ASSIGN SYS013-DA-DK-S.
000026        SELECT ｳﾘｱｹﾞﾆｯｹｲﾋｮｳｰ01
```

```
000555 PROCEDURE  DIVISION.
000556 BOX-001-S   SECTION.
000557     ACCEPT   DSL-DATE  FROM   DATE.
000558     MOVE     DSL-YY          TO        YY-ED.
000559     MOVE     DSL-MM          TO        MM-ED.
000560     MOVE     DSL-DD          TO        DD-ED.
000561     OPEN     OUTPUT    ｳﾘｱｹﾞﾆｯｹｲﾋｮｳｰ01.
000562     INITIALIZE   ｳﾘｱｹﾞﾆｯｹｲﾋｮｳｰ01-RECB.
000563     OPEN     INPUT     ﾄｸｲｻｷｰﾏｽﾀｰｰF1.
000564     MOVE     LOW-VALUE      TO   ﾄｸｲｻｷｰﾏｽﾀｰｰF1-WKEY.
000565     INITIALIZE   ﾄｸｲｻｷｰﾏｽﾀｰｰF1-RECB.
000566     OPEN     INPUT     ｼｮｳﾋﾝｰﾏｽﾀｰｰF2.
000567     MOVE     LOW-VALUE      TO   ｼｮｳﾋﾝｰﾏｽﾀｰｰF2-WKEY.
000568     INITIALIZE   ｼｮｳﾋﾝｰﾏｽﾀｰｰF2-RECB.
000569     OPEN     INPUT     ｳﾘｱｹﾞﾃﾞｰﾀｰI1.
000570     INITIALIZE   ｳﾘｱｹﾞﾃﾞｰﾀｰI1-RECB.
000571     MOVE     SPACE    TO   ｳﾘｱｹﾞｺﾞｳｹｲｰX1.
000572     MOVE     SPACE    TO   ﾆｭｳｷﾝｺﾞｳｹｲｰX1.
000573     MOVE     SPACE    TO   ｳﾘｱｹﾞｼｮｳｹｲｰX1.
000574     MOVE     SPACE    TO   ﾆｭｳｷﾝｼｮｳｹｲｰX1.
000575     MOVE     SPACE    TO   ｴｲｷﾞｮｳｼｮｺｰﾄﾞｰX1.
000576     MOVE     SPACE    TO   ﾃﾝﾋﾟｮｳﾊﾝｺﾞｳｰX1.
000577     MOVE     SPACE    TO   ﾃﾝﾋﾟｮｳｸﾌﾞﾝｰX1.
000578     MOVE     SPACE    TO   ｼｮｳﾋﾝﾒｲｼｮｳｰX1.
000579     MOVE     SPACE    TO   ｳﾘｱｹﾞｷﾝｶﾞｸｰX1.
000580     MOVE     SPACE    TO   ﾆｭｳｷﾝｶﾞｸｰX1.
000581     MOVE     SPACE    TO   ﾄﾘﾋｷﾋﾂﾞｹｰX1.
000582     MOVE     SPACE    TO   ﾄｸｲｻｷﾒｲｼｮｳｰX1.
000583     MOVE     SPACE    TO   ｳﾘｱｹﾞｽｳﾘｮｳｰX1.
000584     MOVE     SPACE    TO   ｳﾘｱｹﾞﾀﾝｶｰX1.
000585     MOVE     SPACE    TO   ﾆｭｳｷﾝｸﾌﾞﾝｰX1.
000586     MOVE     ZERO             TO   DSL-PAGE-COUNT.
000587     MOVE     DSL-FIRST-LINE-COUNT   TO   DSL-LINE-COUNT.
000588     PERFORM  BOX-016-S.
000589     MOVE     ZERO             TO   ｳﾘｱｹﾞｺﾞｳｹｲｰW002.
000590     MOVE     ZERO             TO   ﾆｭｳｷﾝｺﾞｳｹｲｰW003.
000591     PERFORM   ｳﾘｱｹﾞﾃﾞｰﾀｰI1-INP.
000592     PERFORM  BOX-002-S
000593              UNTIL   (ｳﾘｱｹﾞﾃﾞｰﾀｰEOF   =   "Y").
000594     MOVE     ｳﾘｱｹﾞｺﾞｳｹｲｰW002   TO   ｳﾘｱｹﾞｺﾞｳｹｲｰ01.
000595     MOVE     ｳﾘｱｹﾞｺﾞｳｹｲｰ01   TO   ｳﾘｱｹﾞｺﾞｳｹｲｰ01ｰL1.
000596     MOVE     ﾆｭｳｷﾝｺﾞｳｹｲｰW003   TO   ﾆｭｳｷﾝｺﾞｳｹｲｰ01.
000597     MOVE     ﾆｭｳｷﾝｺﾞｳｹｲｰ01   TO   ﾆｭｳｷﾝｺﾞｳｹｲｰ01ｰL1.
000598     IF       DSL-SETFLAG   =      "1"
000599         THEN       PERFORM    BOX-031-S
000600         ELSE       PERFORM    BOX-032-S.
000601     COMPUTE  DSL-LAST-LINE   =
000602              DSL-FIRST-LINE   +     3   -   1.
```

図11 ゼネレーションしたプログラムの例 ゼネレーション結果のCOBOLプログラムについての一部を表わしたものである。

(2) データ構造定義(**図8**参照)
(3) 帳票レイアウト定義(**図9**参照)
(4) 生成式定義(**図10**参照)
(5) ゼネレート結果のCOBOLプログラム(**図11**参照)

6 結 言

ビジネス分野でのソフトウェア生産方式として，データ中心の仕様定義方式とこれを入力とするCOBOLプログラムゼネレータを開発し，この内容について紹介した。

現在，このシステムは具体ユーザーで実験適用中である。この結果，システム設計からテスト完了までの開発工数を約$\frac{1}{2}$に削減できる見通しである。このほか，当システムを用いることによって，(1) 仕様とプログラムの一致化，(2) 仕様記述の標準化，(3) 仕様変更が容易，(4) エンドユーザーによる業務システム開発を推進できるなどの効果が期待できる。今後適用業務の拡大とともに当システムを一貫したソフトウェアの開発環境として発展していきたいと考えている。

最後に，当システム適用に関し貴重な御意見をいただいた適用顧客各位，また共同して開発に協力された関係各位に対し，深く感謝する次第である。

参考文献

1) Prywes, N.S：Automatic Generation of Computer Programs:Advanced in Computers Vol. 16, 1977
2) 鵜飼，外：データ中心型仕様定義を前提としたアプリケーションプログラムゼネレータ——仕様定義方式について：情報処理学会第24回全国大会(昭57-3)
3) 河野，外：同上——プログラムゼネレーション方式について：情報処理学会第24回全国大会(昭57-3)
4) Michael Hammer：A Very High Level Programming Language for Data Processing Applications:Communications of the ACM, November 1977, Vol. 20, Number11
5) M. Jackson：構造的プログラム設計の原理：日本コンピュータ協会
6) 河野，外：データ中心型仕様定義を前提としたアプリケーションプログラムの生成方式:情報処理学会ソフトウエア工学研究会資料26(昭57-10)
7) 鵜飼，外：同上——仕様定義手順について:情報処理学会第25回全国大会(昭57-10)
8) Ken Orr：Structured Requirements Definition:Ken Orr and Associates, Inc.