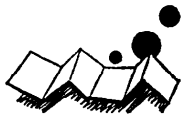
解　説

3. 応用分野の最前線



3.3 事務処理分野における自動プログラミング†

原　田　実††

1. はじめに

情報化社会が進展し，ビジネスの機械化が急速な勢いで進んでいる．企業はより良い質の情報をより速く得るために，膨大なソフトウェアを作り続けている．その作成現場では，活動のほとんどが人手に頼って行われているのが実状で，プログラム作成に手間がかかり過ぎる，作成されたプログラムが仕様書どおりでない，仕様書とプログラムの両者を矛盾なく維持管理するのが難しいなどの問題を抱えて四苦八苦しているありさまである．これはソフトウェア生産が他の産業に比べ自動化が大変遅れているからであり，本質的に解決するには他の工業製品と同様にプログラムを自動的に作成するしかないように思われる．この章では，このような問題を解決する手段としての自動プログラミング[1]の考え方について述べる．

1.1 開発者インタフェースの高度化

プログラムの作成がすべて手作りというわけではない．機械語コードをプログラムと捉えるなら，多くの高級言語にはコンパイラがあって機械語への変換は自動的に行われる．実際コンパイラのことを自動プログラミングといった時代もあった．したがって自動プログラミングの歴史は，図-1 に示すように人間がコンピュータに与える指令（プログラム）を開発するインタフェースの高度化の歴史といえる．このスペクトル上で考えると，本解説で扱う自動プログラミングの多くは，COBOL で記述したようないわゆるプログラムよりもより人間向きなインタフェースから機械語に自動的に変換可能な低レベルのインタフェース（多くの場合は COBOL プログラム）へ変換する技術として捉えることができる[7]．したがって，直接計算結果を返すような自動再計算機能付き表言語とか自然語インタフェースによるデータベースへの問合せ言語などの対話的応答システムは取り上げない．

このようにしてプログラムの**開発者インタフェース**が人間よりになり，機械語への変換が自動化されれば，プログラムの生産性も信頼性も向上し，途中のインタフェース（いわゆるプログラム）を管理する必要もなくなり，先にあげた諸問題も解決すると期待するわけである．

1.2 プログラミング作業の自動化

開発者インタフェースが人間よりになるということは，機械語までの変換過程で従来人手で行われていた作業が自動化されることを意味する．この自動化される作業こそが通常われわれがプログラミングと呼んでいる作業である．したがって，事務処理分野における自動プログラミングを，事務処理アプリケーションソフトウェアを作成する際のプログラミング作業を自動化する技術の総称と考える．言い換えれば，自動プログラミングシステムを，専門的なプログラマがもつプログラミング知識を蓄えこれを自動的に適用するエキスパートシステムと捉えることもできる．ではこのプログラミング作業とは具体的にはどんな作業であろうか．

B. W. Boehm によるソフトウェアのライフサイクルモデルによれば，ソフトウェア開発における生産工程は要求分析，概要設計（彼は製品設計と呼んでいる），詳細設計，コーディング／デバッギング，インテグレーションからなり，プログラミングという工程はない[3]．これらの工程で一般的に行われている作業を整理すると表-1 のようになる．しかし，ソフトウェア開発に関するコスト評価モデル COCOMO（構成的コストモデル）の定義に当たって，Boehm は詳細設計とコーディング／デバッギングをまとめて**プログラミング**としている．すなわち，プログラミングには，ソフトウェアの構造設計（モジュールへの分割）

† Automatic Programming in Business Application by Minoru HARADA (Central Research Institute of Electric Power Industry: CRIEPI).

†† (財)電力中央研究所知識処理研究室

は含まないが，各モジュールに対する解の基本的なアルゴリズムの決定は含むものと考えている．

1.3 入力仕様はどうあるべきか

Balzer は，図-2 に示すように，自動プログラミングシステムを高レベルの仕様を獲得する機能と，これを自動的に機械語にコンパイル可能な低レベルの仕様に自動的に変換する機能をもつシステムと捉えている[2),15)]．前節の考えに立てば高レベル仕様は概要設計書であり，低レベル仕様はプログラムに当たる．

しかし，仕様獲得機能を仕様理解機能と発展させて捉えれば，要求仕様といっても良いくらいのかなり人間向きな仕様を入力としても良い．ではこのような人間向きの高レベル仕様とはどのようなものであろうか．

これまで，開発者インタフェースと呼んだり，概要設計書と捉えたり，要求仕様にまで拡大解釈した，この自動プログラミングシステムの入力となる高レベル仕様を，以降では**プログラム仕様**（誤解のないときは単に**仕様**）と呼ぶ．このプログラム仕様の記述方法が，どの程度簡潔で，理解しやすく，記述しやすいものであるかが，このシステムの利用性を大きく左右する．以下に，プログラム仕様がもつべき望ましい条件をい

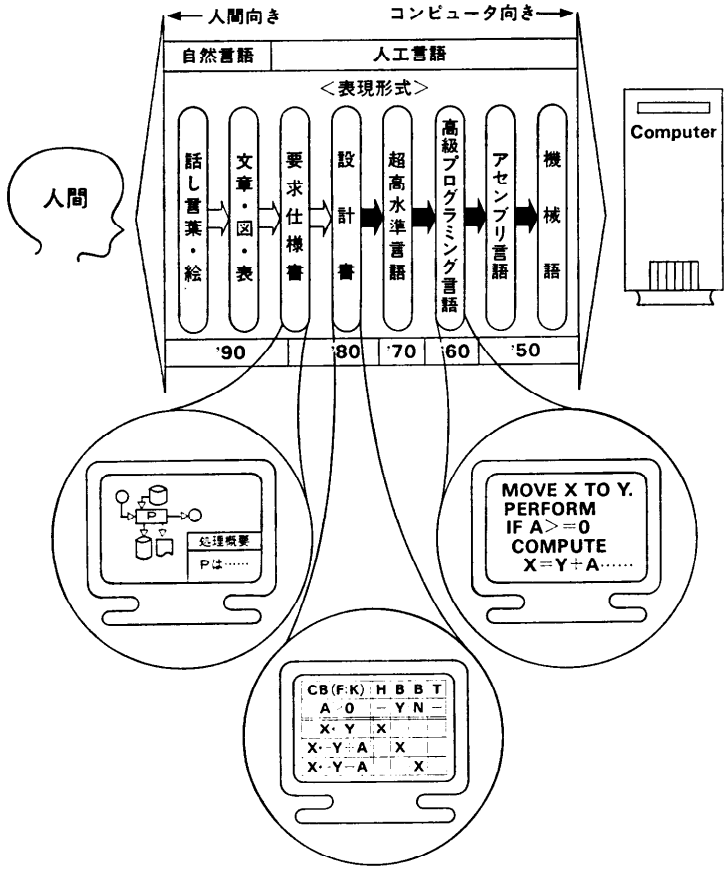


図-1 開発者インタフェースの高度化

表-1 ソフトウェア・ライフサイクルにおける生産工程

工　程	生産活動の詳細	類似概念
要求分析	ソフトウェアに対する要求を分析し，求められる機能，インタフェース，性能を正しく反映した完全な要求仕様書を作成する．	要求仕様化
概要設計	要求仕様書を満足するソフトウェアを作成するために必要な資源や環境を決定し，ソフトウェアをモジュールに分割し，その機能やインタフェース，それらの間の呼出し関係，使用するデータの構造などを記述した完全な概要設計書を作成する．マニュアルやテスト計画書の作成も行う．	機能設計 構造設計 初期設計 基本設計 製品設計
詳細設計	与えられた概要設計書に示された要求を満たすために，モジュールがどのようにデータを処理するかの基本的アルゴリズムを詳細に記述した詳細設計書を作成する．	論理設計 手順設計 プログラム設計
コーディング／デバッギング	モジュールの詳細設計書に基づいて，その正しいコードを特定のプログラミング言語を用いて作成する．モジュールテストも含まれる．	製作 インプリメンテーション
インテグレーション	個々のモジュールを組み合わせて，要求されるソフトウェアを組み立てる．統合テストが中心となる．	総合テスト

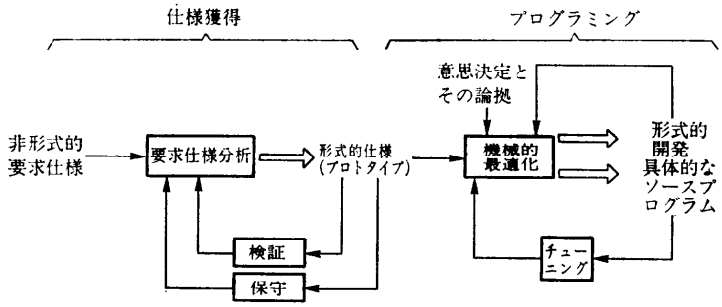


図-2　自動プログラミングシステムのモデル[1]

くつか列挙する．なお，以降でプログラム仕様の構成要素を（プログラム命令同様，それを個々の問題に特化するための関連するパラメータや引数なども含めて）**仕様要素**と呼ぶ．

［**抽象化**］ 事務処理全般にあるいは特定の業務に共通して良く現れる**プログラムロジック**（プログラムが行う詳細なデータの処理手順）を，個々の仕様要素1つで指定できる．これによって，プログラムロジックを構成するための論理的な思考をいくらか省略できる．

［**非手続き的**］ 仕様要素をその順序を気にせず組み合わせることによって，プログラム仕様を構成できる．これはシステムが仕様要素間の正しい関係を推論するからであり，その分利用者は求めるプログラムにおける計算式や条件を思いつくままに指定できる．

［**最適化**］ 特別の指示をプログラム仕様に記述しなくても，できるだけ速くて省資源のプログラムを生成できる．これはプログラム仕様の条件というよりも自動プログラミングシステムの重要な機能である．特に事務処理ではファイルのアクセスを減らすこと，できるだけ順編成ファイルを使うことなどが指針となる．

［**視覚化**］ 表現方法が視覚的である．プログラムは従来からそのほとんどが1次元のテキスト形式言語であり仕様要素を関係付ける手段に隣接関係しかなく表現力が乏しい．図や表のような2次元の記述方法で仕様を表すなら，仕様要素である図形や語（ときには文のこともあるが）はその接続関係や記入場所によってさまざまな意味で明確に関係付けられる．この結果，仕様が視覚化され，さらに内容を正確かつ簡潔に表現・理解できるようになり，利用者はプログラムで記述するよりも目的とするシステムの諸特性に対するはるかに良い理解を得ることができる．

［**モデル化**］ 生成されたプログラムが特定の明確なモデル（以後，**プログラムモデル**と呼ぶ）に従ってデータを処理する．この結果求める機能を果たすプログラム仕様を作成するために，どのように仕様要素を組み合わせれば良いかに関する判断基準が得られ，仕様を作成しやすくなる．

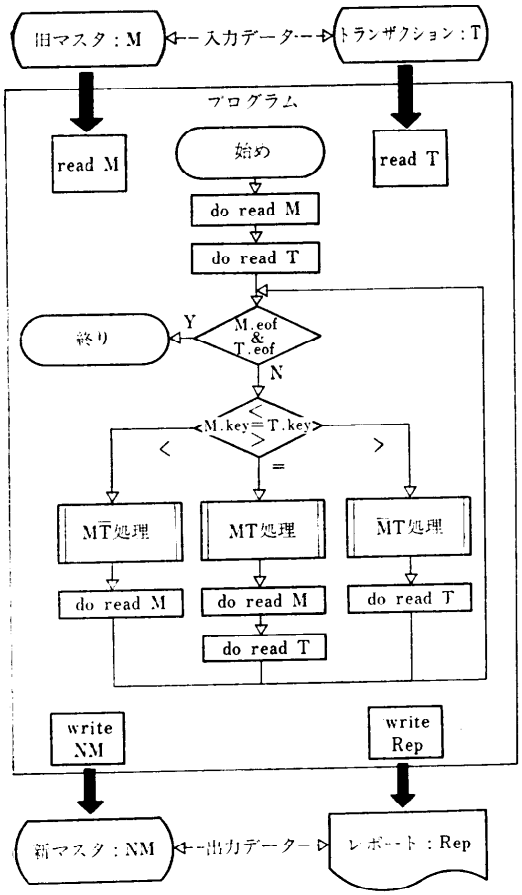
これらの性質のうち，モデル化や抽象化はプログラム仕様に形式性を要求する．一方視覚化や非手続き化は非形式性を要求する．これらの相反する性質をどのように交ぜ合わせたり，重ね合わせたりして利用しやすいプログラム仕様を構築できるかが自動プログラミングシステムの成否を左右すると思われる．

2. 事務処理プログラムの特性とコード生成方式

2.1 計算ロジックと制御ロジック

一般に，事務処理プログラムは本質的にたちの悪い問題（wicked problem）といわれ，プログラムの出力応答がどのようにしてその入力から得られるかの手順や構造を与えないとその仕様を記述できないといわれている．これは，たとえば技術計算プログラムの仕様を，解くべき方程式とその精度によって宣言的に与えることができるのと対照的である．これは事務処理が，従来手順的に行われていた伝票処理などの事務計算を機械化するためのプログラムであることに起因している．

さて，このような事務処理プログラムは，扱うデータが外部装置にありまたその結果を人間に理解しやすいように整理して出力するために，単に計算を行うだけでなくデータの入出力制御も行う必要がある．したがって，そのロジックは，図-3 に示すように，**制御ロジックと計算ロジック**に分けられる．前者は入力データの構造に依存してデータをレコード単位でファイルなどからプログラムに取り込み，計算結果を出力データの構造に依存してレコード単位でプリンタなどに送り出す方法を主として定めている．後者は出力データの意味的性質に依存して入力データの加工方法を主として定めている．たとえば，図-3 は2本のファイルのマッチング処理を表しているが，計算ロジックはMT 処理，MT 処理，MT 処理の中に記入する計算式群（write NM や write Rep の起動命令を含



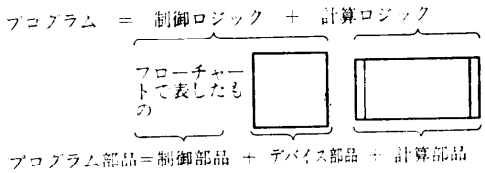


図-3 プログラム＝制御ロジック＋計算ロジック

む）であり，それ以外のフローチャートで表したもの（read M や write NM の詳細を含む）は制御ロジックである．

2.2 プログラムの部品化

事務処理プログラムでは，制御ロジックの割合が他の分野に比べ大きい．また計算ロジックは問題ごとに異なるため利用者が与えざるをえないが，制御ロジックはレコード単位の入出力処理を表しており，扱うデータの構造に依存するため，いくつかの型（制御パターン）に分類できる．たとえばファイル処理を行うバッチプログラムでは，図-4 に示すように以下の3つの制御パターンに分類できる．

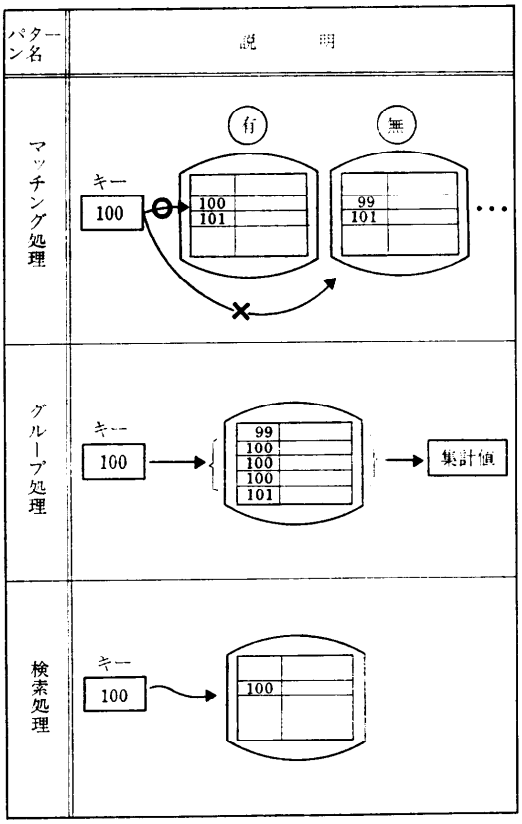


図-4 バッチ型プログラムの3つの制御部品

①[マッチング処理] 複数のファイルからレコードを入力し，キー項目の値の組合せに応じて特定の処理を行う．

②[グループ処理] ファイルからレコードを順に入力し，キー項目が同一値をもつレコード群ごとに特定の処理を行う．

③[検索処理] キー値を指定して，ファイルから1レコードを直接取り出して処理する．

これらのパターン化された制御ロジックを，ファイル名やキー項目名などをパラメータ化してコーディングした汎用プログラムは，標準処理パターン，雛型プログラム，プログラムスケルトンなどと呼ばれ，事務処理プログラムの手作業による開発において広く再利用されている．このような再利用過程を自動化して自動プログラミングシステムを構築する場合には，これ

らの汎用プログラムは**プログラム部品**として用いられる．

プログラム部品は図-3 に示したように，制御部品，デバイス部品，計算部品の3つに分けることができる．

デバイス部品は，図-3 の read M や write M のように，ファイルなどのデバイスの特性に依存した入出力手続きを部品化したものである．**制御部品**と**計算部品**はそれぞれ制御ロジックおよび計算ロジックからデバイス部品を除いて部品化したものである．

制御部品は制御パターンを基に，たとえば2本マッチング部品，3本マッチング部品…などとして，容易に部品化できる．デバイス部品も，たとえば順ファイル単純入力部品，順ファイル2段バッファリング付き入力部品などとして，容易に部品化できる．これに対し計算部品は，計算ロジックが本来業務に依存するので，西暦和暦変換やコード系変換などを除けば汎用の部品は作成しにくく，作成しても利用頻度は低い．

2.3 コード生成方式

事務処理プログラムを作成するプログラミング技術を凡例コーディング（プログラム部品）としてあるいはプログラミング規則として知識化し自動再利用しているかで，事務処理分野のほとんどの自動プログラミングシステムのコード生成方法は以下の3つに分けられる．

①**[プログラミング方式]** プログラム仕様に，扱うデータの構造を指定し，これにジャクソン法やワーニエ法[42]などのプログラミング技法を適用して，制御ロジックを生成する．

②**[部品合成方式]** プログラム仕様にプログラムの制御パターンを示す情報を指定し，それを基に適切な制御部品を検索し，その汎用部分をプログラム仕様に合わせて特化し，制御ロジックを生成する．

③**[コンパイラ方式]** プログラム仕様の各仕様要素ごとにその意味解釈として対応するコードを生成する規則があり，これを特定のモデルに従って適用することで制御ロジックと計算ロジックを生成する．

このうち①と②では，生成されたプログラムの骨格を表す制御ロジックに，問題に固有の計算ロジックを組み込む必要がある．先に述べたように制御ロジックはパターン化できるので，計算ロジックを組み込む場所は，制御ロジックに基づいてプログラムがレコードを処理する相違なる状態―たとえば，2本マッチングでは，マスタのみレコードあり（例：図-3 の MT 処理），トランのみあり，両方にありの3状態―を表すラベルなどで表現される．以後このようなラベルを制御ロジックあるいは制御部品の**処理状態ラベル**と呼ぶ．したがって①や②では処理状態ラベルに対応して利用者が直接 COBOL などの言語で計算ロジックを記述すれば―**オウンコーディング**という―，プログラム仕様が完成する．

一方③では，制御ロジックも計算ロジックもともに，与えられた仕様要素を用いて統一的に記述する．しかしこのときも多くのシステムにおいて，仕様要素は制御ロジックを中心に定められ，計算ロジックはそのパラメータとして記述することが多い．

また，②と③はともにプログラム部品を用いてコードを生成するが，部品の規模とその扱いが大きく異なる．すなわち，②では1部品で制御ロジックをカバーするのに，③では数部品でカバーする．また②では部品は凡例コードとして存在するのに対し，③ではコードの生成手続きとして存在する．したがって③では，モデルによっては，柔軟に部品（仕様要素）を組み合わせることによって，多様な制御パターンをもつ制御ロジックを容易に構成できる．

これらの自動化の仕掛けについては本特集号の他の解説記事で詳しく説明されている．したがって本解説で紹介する各システムについては，主としてプログラム仕様がどのように高度化されているか，その本質は何かなどについて詳しく述べ，コードの生成方式については上のどの方式によるかを述べる程度に留める．

以下では，事務処理用の自動プログラミングシステムを，目的や機能の点から，部品自動合成ツール，超高水準言語，仕様コンパイラ，要求コンパイラに分けて説明する．なお，技術的に同レベルであれば，研究中のものより実利用されているものを主として説明する．

3. 部品自動合成ツール

部品合成方式による自動プログラミングシステムで，部品の結合方法や特化（カスタマイズ）方法を指定するパラメータを**メニュ方式**や**マクロ展開言語**を用いて入力し，これに合わせて適切なプログラム部品を検索し，編集し，結合してプログラム全体を生成する．したがってこれらのシステムを特徴付けるのはパラメータの与え方である．なお，利用できる部品がないとき，適用性が急激に低下することが共通の欠点である．

3.1 階層メニュ画面によるもの

プログラム仕様を階層化されたメニュ画面で与えるもので，パラメータは表示されたメニュから選択する．例として，IPA（情報処理振興事業協会）の古宮らによる PAPS[20]，キヤノンソフトウェアの CANO-AID[12] などがある．

3.1.1 PAPS

PAPS の開発者インタフェースは階層化されたメニュ画面であり，利用者はプログラムごとにその骨格を構成すると思われる制御部品名と，それが扱う入出力ファイルの属性を指定する．ファイルの属性に従って，レコードを入力するデバイス部品が検索され組み込まれる．計算ロジックの指定は現在はオウンコーディングだが，誘導型のデシジョンテーブルを用いて高度化することを計画している．

3.1.2 CANO-AID

CANO-AID では，プログラム部品を，計算ロジックを記述した計算部品，DB/DC インタフェースを記述した標準部品（デバイス部品のこと），制御部品としての汎用部品の3つに分けて名前を与えて管理する．

利用者はまず業務に現れる計算ロジックを洗い出し，それらをオウンコーディングし名前を与えて計算部品とする．次に，プログラムごとにその主制御を表す汎用部品名を指定すると，その汎用部品が必要とする計算部品や標準部品を入力するメニュ画面が表示されるので，これに適切な部品名を入れるとプログラムが生成される．

3.2 マクロ展開言語によるもの

プログラム仕様記述にマクロ展開言語を用いるもの．たとえば，日本電子計算の PRECOBOL[31]，日立ソフトウェアエンジニアリングの STAMPS[16]，日本システムサイエンスの JASMAC[40]，東芝の SPC-LAN[39]，富士通の MASCOT[29]，Olson Research Associates 社の Rice らによる ASGS[34] などがある．

3.2.1 PRECOBOL

```
01  N2R12MI-WREC          COPY   'N2RL012'     SUPPRESS.
01  N2R03MO-WREC          COPY   'N2RL003'     SUPPRESS.
#END-DATA.
PROCEDURE          DIVISION.
CONTROL-SECT               SECTION.
CONTROL-RTN.
    #ENTRY   100-INITIAL.        ┐
    #ENTRY   200-SHORI.          ├①
    PERFORM  XKEN-OUT.           │
    STOP     RUN.                ┘
CONTROL-EXT.
    EXIT.

    @ID(PROG=UP2050,AUTHOR=KOTA,DATE=62.01.22)
    @ENV
    @FD/N2R12MI    (KIND=DSK,BLOCK=(CHARA=1120,REC=0))
    @FD/N2R03MO    (KIND=DSK,BLOCK=(CHARA=250, REC=24))
    #DATA.
    77  IX                        PIC    9(04)  VALUE  ZERO.
    77  IDX                       PIC    9(04)  VALUE  ZERO.
    77  SELECT-FLG                PIC    9(01)  VALUE  ZERO.
    01  WORK-AREA/WK-.
        03 SETAI.
             05  NEW-KEY              PIC     X(07).
             05  FILLER               PIC     X(03).
        03  OLD-KEY                   PIC     X(07).
#BOX    200-SHORI.        ←②
    @BASE-BRK/200-SHORI
      ③ (OPEN   =  "OPEN   INPUT  N2R12MI.          ┐⑤
                    OPEN   OUTPUT  N2R03MO."        ┘
       ,AFILE =  210-N2R12MI
       ,LEVEL =((A-KEY = WK-NEW-KEY                 ┐
                ,W-KEY = WK-OLD-KEY                 │
      ④         ,RESET = "#ENTRY 220-RESET."        ├⑥
                ,PROC  = "#ENTRY 230-HENSHU."       │
                ))                                  ┘
       ,DET    =   "#ENTRY240-TABLE-STORE."         ┐
       ,CLOSE  =   "CLOSE  N2R12MI.                 ├⑦
                    CLOSE  N2R03MO."                ┘
      ).
#BOX    210-N2R12MI-READ.
    @GET-SIMPLE/210-N2R12MI
       (AFILE =  N2R12MI
       ,INTO  =  N2R12MI-WREC
       ,POST  =  "ADD   1        TO     XKEN (01).
                  MOVE  M12-SETAI TO     WK-SETAI."
       ).
```

図-5　PRECOBOL によるプログラム仕様記述例

PRECOBOL は，プログラム仕様を図-5 に示すような方法で記述するマクロ言語である．この言語はモジュールの実行順序①と，各モジュール②の基になるプログラム部品名③とその汎化パラメータ④の並びで記述される．

その特徴は，汎化パラメータの実パラメータとして，⑤，⑥，⑦のように計算ロジックを指定できることである．プログラム部品には，あらかじめ提供されているもののほかに，部品記述言語 GCL を使って，利用者が作成するものもある．部品を作成するには，まず部品が行うべき処理の概要を記述し，その中で問題ごとに変わり

そうな部分を切り出し仮パラメータとして指定する．

汎化パラメータはプログラム部品内の制御構造に対応した構造をもつことができるので，1つの部品を多段のグループ処理に用いることができる．

さらに GCL で記述した部品に新しいパラメータを付け加えても，そのパラメータを用いていない既存のプログラム仕様からのプログラム生成に影響を与えないようになっている．

3.2.2 その他

JASMAC, STAMPS, SPC-LAN, MASCOT は PRECOBOL とほぼ同様のシステムであるが，部品を利用者が定義できるかどうか，部品を特化するためのパラメータの与え方や実パラメータとして許される内容などによって特徴がある．

```
ID      REI101.
SA      売り上げデータ          FILE(U01)     RECORD(売り上げレコード).
SA      正しい売り上げデータ    WORK          RECORD(売り上げレコード).
SA      不良データ              WORK          RECORD(売り上げレコード).
SA      売り上げ累計マスタ      FILE(U02)     RECORD(売り上げレコード).
SA      新売り上げ累計マスタ    FILE(U03)     RECORD(売り上げレコード).
SA      不良データ一覧表        FILE(U04).
RECORD.
  01    売り上げレコード・
        02  商品コード         PIC 9(2).   }
        02  売り上げ数量       PIC 9(6).   } ②
        02  売り上げ金額       PIC 9(8).   }
① CHECK     IN(売り上げデータ)      OUT(正しい売り上げデータ，不良データ).
③ MATCH     IN(売り上げ累計マスタ，     正しい売り上げデータ)
                                    OUT(新売り上げ累計マスタ)
            KEY(商品コード).
            UPDATE  INSERT          SUM(売り上げ数量，売り上げ金額).
④ REPORT    IN(不良データ)          OUT(不良データ一覧表)
            RD(不良データレポート).
RD      不良データレポート          STANDARD(NOAUTOEDIT).
SOURCE.
  PH        NC″ ＊＊＊  不良データ一覧表  ＊＊＊″.
```

図-6 HYPER COBOL によるプログラム仕様記述例[7]

ASGS はスケルトンといわれる標準プログラムを RSL という言語で記述したプログラム仕様に合わせて編集したプログラムを生成するツールである．ただし，Rice の研究はプログラム自動生成システムの一般的な構成や構築法に中心がある．

4. 超高水準言語 (Very High Level Language: VHLL)

ある目的を果たすために事務処理プログラムで多く用いられるロジックをユニット（マクロ）化したものを言語命令としてもち，これを要求に適合化させるパラメータとともにつなぎあわせてプログラム（仕様）を構成する超高水準なプログラミング言語であり，第4世代言語，問題向き言語，簡易言語などとも呼ばれる．

3.2 であげたマクロ展開言語を開発者インタフェースにもつ部品自動合成ツールとの違いは，第1に超高水準言語は明確なプログラムモデルをもちそのモデルに従って仕様要素を結合するのでプログラム仕様を作成しやすいということ，第2は超高水準言語は主としてコンパイラ方式あるいはプログラミング方式でプログラムを生成するのに対し，部品自動合成ツールでは部品の自動編集・結合によってプログラムを生成するということである．

現在用いられている超高水準言語のモデルは**データフローモデル**と**データ構造駆動モデル**が多い．

4.1 データフローモデルによるもの

処理要素間をデータが流れ，加工され，最終的に結果が得られるというモデルに基づいて記述されたプログラム仕様から，処理要素ごとにその意味を表すコードをコンパイラ方式で生成する．このときデータはデータフローの理論に従って1レコードずつ流れる．

例としては，富士通の HYPER COBOL[7),8)]，三井造船の DF-COBOL[7)]，日本電気の IDL[8)]，筑波大の久世らによる Stella[21)] などがある．

4.1.1 HYPER COBOL

HYPER COBOL は DF-COBOL を拡張して開発されたもので，処理要素はユニット，データフローはストリームエリヤで表される．ユニットは複数の入出力ストリームエリヤをもち，各ストリームエリヤには高々1レコードのデータを保持することができる．ユニットはそのすべての入力ストリームエリヤにレコードが揃い，かつ出力ストリームエリヤにレコードが溜まっていないときのみ起動する．ユニットが起動された後は，入力ストリームエリヤのレコードは消費され，ユニットごとに記述された処理条件に従って入力データを処理した結果に依存して決まる出力ストリームエリヤに1レコード出力される．このユニットの起

動は論理的には並列であり，同時に複数のユニットが起動状態にあることもある．

ユニットにはマッチング処理を行う MATCH やグループ処理を行う SUMMARIZE など 12 のシステム提供ユニットがあり，このほかに CONTROL タイプユニットとして利用者側でユニットを定義することができる．

図-6 に HYPER COBOL で記述したプログラム仕様の例を示す．ここでは，①の CHECK ユニットで売上データが②に示した属性どおりのデータかどうか検査し，③の MATCH ユニットで正しい売上データと売上累計マスタをマッチングして新売上累計マスタを作成することとともに，④の REPORT ユニットで不良データを標準形式でリスト出力している．

HYPER COBOL はデータフローの概念を徹底的に踏襲しているために，モデルとしては美しいのであるが，半面 SORT ユニットをプログラム仕様中に1度しか使えないとかオンライン処理用のユニットを定義できないなどの問題を抱えている．

4.1.2 その他

IDL では，データフローは中間ファイルとして実現され，各ユニットは起動されると，その入力ファイルのレコードをすべて処理し終わるまで制御を保持し，出力ファイルにレコードを書き出す．したがって，オンラインやソート処理も自由に記述できる．

Stella は，Pascal ベースのデータフロー型言語で，ストリームを流れるデータとしては Pascal で記述可能なデータ構造をもつデータを指定できる．ユニットはすべて利用者が定義し，next 命令を使って，ストリームからデータを入力したりそこに出力する．

4.2 データ構造駆動モデルによるもの

入出力データの構造を指定し，これを基に制御ロジックを生成するシステムであり，計算ロジックの与え方に個々のシステムの特徴が現れる．

たとえば，住商コンピュータサービスの ALICE[44]，日本システムサイエンスの JASPOL[8]，ペンシルベニア大の N. S. Prywes らによる MODEL[33] などがある．

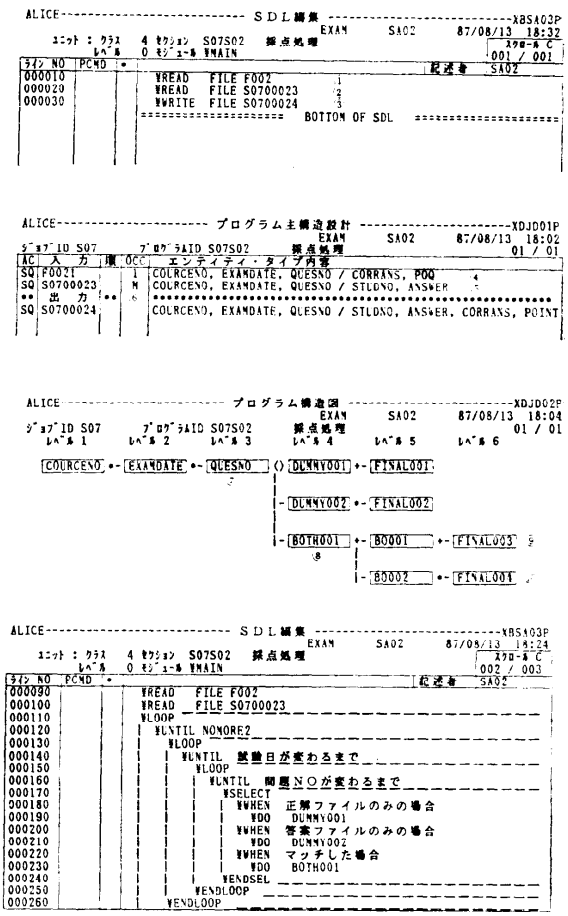


図-7 ALICE によるプログラム仕様入力例

4.2.1 ALICE

ALICE は VHLL というよりも，システム開発を設計からコーディングまで対話的に支援する半自動化ツールである．ただしその過程で一貫して，SDL といわれる言語を用いてシステムの特性を記述する点，および入出力データ構造からジャクソン法により制御ロジックを自動生成する点で，ここに分類した．

図-7 は試験結果の採点処理を行うプログラムを ALICE を用いて開発する段階を示している．まず利用者は，(a)に示すように，このプログラムが正解ファイル①と答案ファイル②を入力し，採点ファイル③を出力することを指定する．するとシステムは，これらのファイルの構造をデータディクショナリから検索し，(b)の④，⑤に示すように表示する．この例では，正解ファイルと答案ファイルはともにそのレコー

```
<SUMM>.
   PGID(マッチング).
<FILE>.
IN-マスタ.
   ASSIGN(U01),  IN, FTYP(SEQ).
トラン.
   ASSIGN(U02),  IN, FTYP(SEQ).
NEW-マスタ.
   ASSIGN(U03), OUT, FTYP(SEQ).
<DATA>.
IN-マスタ.
01 IN-REC.
   02 I-商品コード 9(4) <L1>.----------①
   02 I-単価       9(4).
   02 I-売上金額   9(8).
トラン.
01 TR-REC.
   02 T-商品コード 9(4) <L1>.----------①
   02 T-数量       9(4).
NEW-マスタ.
01 NEW-REC.   <TMNG(L1)>.-----------②
   02 I-商品コード 9(4).
   02 I-単価       9(4).
   02 W-売上金額   9(8).
<DET.トラン>.-----------③
   ¥¥COND(IN-マスタ(OFF)), ELSE-GOTO(正常処理).
     DISPLAY "トランのみ  T-商品コード=" T-商品コード.
     ¥¥EXIT.
正常処理.
   ADD T-数量 TO W-数量.
<L1>.-----------④
   COMPUTE W-売上金額 = W-数量 * I-単価 + I-売上金額 .
<WORK>.
77 W-数量       9(4) ZERO <RESET(L1)>.----⑤
77 W-売上金額   9(4) ZERO <RESET(L1)>.----⑤
<END>.
```

図-8 JASPOL によるプログラム仕様記述例

ドが，コース No.，試験日，問題 No. をキーとしてこの順にソートされ，おのおの正解と配点および学生 No. と解答を従属項目としてもつこと，さらに答案ファイルには同一キー値をもつレコードが複数（⑥の M）あることが示されている．さらにシステムはこのデータ構造を基に制御ロジックを(c)に示すように導く（図中，＊は繰返し，＜ ＞は選択，＋は順次構造を表す）．たとえば，問題処理⑦は3つの処理の選択からなり，その1つである“両ファイルにレコードあり処理” ⑧は“正解ファイルの読込み処理” ⑨と“答案ファイル読込み処理”の繰返し⑩を含むことなどが表現されている．

このような制御構造を見ながら，システムが問い合わせる繰返し条件や選択条件に答えると，最終的に(d)に示すような制御ロジックが生成される．後は，この中で ¥DO の所に計算ロジックを SDL で記述すると，最終的に COBOL プログラムが生成される．

4.2.2 JASPOL

JASPOL では，データ構造定義に計算の実行タイミングを指定する〈Ln〉などの処理状態ラベルを割り当てる．システムはワーニエ法を基に生成された制御ロジックの処理状態ラベルで指定されるところに，このラベルごとに記述された計算を割り付けることによってプログラムを生成する．

ワーニエ法[42]では設計の初期段階で，すべての順ファイルを同一キー値をもつレコードでまとめてロジカルインプットファイルを作成するが，JASPOL はこれと同等のものをマージドストリームという論理入力ファイルとしてシステムが仮想的に生成する．利用者は Ln で指定されるキーの各繰返しレベルごとに具体的な計算ロジックを与えることで，プログラム仕様を記述できる．

たとえば，図-8 は売上データを集計しながらマスタとマッチングし，新マスタを作る処理を JASPOL で記述したプログラム仕様例である．①は，I-商品コードと T-商品コードを対応するマッチングキーとしてマージドストリームを作成し，このレベルを L1 とすることを指定している．②は新マスタレコードを L1 のキーが変わったときに出力することを指定している．③はトランというファイルからきたマージドストリーム内の各レコードに対する処理を以下に書けというラベルである．ここでは，¥ L1 COND 文を用いて，マスタレコードがないとき (IN-マスタ (OFF)) は次行の DISPLAY 命令でエラー処理を行い，そうでないときは“正常処理”ラベルに飛び，そこで T-数量の加算を行う．④には，L1 レベルのキーが変わったときに行う処理として，W-売上金額の計算式を書く．⑤は，作業データである W-数量と W-売上金額がともに L1 が変わるとリセットされることを示している．

このように JASPOL はワーニエ法によるプログラミング技法をマージドストリームとしてモデル化したプログラミング方式の自動プログラミングシステムといえる．

4.2.3 MODEL

MODEL では非手続き的にプログラム仕様を与えることに重点が置かれており，これからデータの依存関係を推論し，それを基に手続き的な言語である PL/I および COBOL で記述されたプログラムを生成する．このためプログラム仕様では，入出力データ構造をワーニエ的な方法で指定し，出力データと入力データの関係は代入式ではなく等式として記述する．

たとえば，図-9 に売上ファイル (IN) と製品ファイル (ITEMS) を入力して総売上量 (TOTAL) と売上高 (COST) を記入した売上レポート (OUT) をリストするプログラム仕様の MODEL による記述例を示す．

```
/* HEADER */
MODULE: SALES;
SOURCE: IN, ITEMS;
TARGET: OUT;
/* DATA DESCRPTIION OF SOURCE AND TARGET FILES */
1 IN IS FILE
  2 INGRP (*) IS GROUP,
    3 INREC (*) IS RECORD,                    } ①
      4 ITEM IS FIELD (PIC '(6)9')
      4 QUANT IS FIELD (PIC '(6)9');
1 ITEMS IS FILE KEY IS ITEM ORIG IS ISAM, ②
  2 ITEMREC IS RECORD,
    3 ITEM IS FIELD (PIC '(6)9'),
    3 PRICE IS FIELD (DEC (6,2));
1 OUT IS FILE,
  2 OUTREC (*) IS RECORD,
    3 ITEM IS FIELD (PIC '(6)9'),
    3 TOTAL IS FIELD (PIC 'B(6)9.V99'),
    3 COST IS FIELD (PIC 'B(6)9.V99');
  /* EQUATIONS DEFINING DATA STRUCTURE PARAMETERS */
③/* a1 */POINTER.ITEMREC=IF END.INREC(SUB1) THEN IN.ITEM(SUB1);
④/* a2 */ END. INREC = (IN.ITEM ^= NEXT. IN. ITEM);
  /* EQUATIONS DEFINING TARGET VARIABLES */
⑤/* a4 */ OUT. ITEM = ITEMS.ITEM;
⑥/* a5 */ TOTAL = SUM (QUANT(SUB1), SUB1);
⑦/* a6 */ COST = PRICE*TOTAL;
```

図-9 MODEL によるプログラム仕様記述例[29]

まず，入出力データ構造を与える．①は，売上ファイルが製品グループ (INGRP) の不定個（＊でこのことを示す）の繰返しからなり，さらに製品グループは製品レコード (INREC) の不定個の繰返しからなることを示している．言い換えれば，売上ファイルは製品 (ITEM) ごとにソートされており，同一製品のレコードが複数件ありうることになる．また②は，製品ファイルが製品 (ITEM) をキーにしてその価格 (PRICE) を与える索引ファイルであることを示している．

MODEL では，このようなデータの階層構造を多次元配列構造として捉え，各配列要素がその上位配列要素のどの位置にいるかを指定するために END, POINTER, NEXT などの前置子が用意されているる．

次に，これらを用いて，データ構造の意味的性質を表現する．③は，INGRP の第 SUB 1 番目の下位要素である INREC が INGRP 内の最後の要素である（論理変数 END. INREC が真）時，索引ファイルの検索キー項目 (POINTER. ITEMREC) が IN. ITEM(SUB 1) と等しいことを示す．④は INREC が INGRP 内の最後の要素であるということが，現在の INREC の IN. ITEM と次の INREC の IN. ITEM（これを NEXT. IN. ITEM で表す）が等しくないという事実と同等であることを示している．

最後に，出力データがどのようにして入力データから求まるかの依存関係を等式を用いて与える．たとえば，⑤は出力の ITEM が入力の ITEM に等しいことを，⑥は出力の TOTAL が INGRP 内の（インデックス SUB 1 が動く範囲内においての）すべての INREC の売上数量 (QUANT) の総和に等しいことを，⑦は出力の COST が ITEMS の PRICE と TOTAL の積に等しいことを，おのおの指定している．

MODEL はこのような非手続き的なプログラム仕様に含まれるデータ間の依存関係を配列グラフに展開し，それをいくつかの極大強連結成分に分解し，その結果を位相ソートし，さらに繰返し処理のスコープが極大になるように最適化した後，プログラムコードに変換する．生成されたプログラムにおいては，レコードはその構成項目が引用されている等式を評価する時点で入力され，すべての構成項目の値が求まった時点で出力される．

MODEL はプログラムの生成以外に，プログラム仕様を理解する段階で，名前の使い方に関するあいまいさ，変数の未定義による不完全性，インデックスの範囲不整合や循環的定義による矛盾などの仕様中の誤りも検出する．

4.3 その他のモデルによるもの

IPA の西村（現日本電気）らによる MOTHER SYSTEM[30] は出力コードの各項目ごとにその値を求める条件と計算式を記述することによって，プログラム仕様を非手続き的に記述しようとした VHLL である．生成されるプログラムは主ファイルからレコードを読みながら，指定された条件に従って関連ファイルを読み込み，出力レコードの各値を求めていく．ただし，1つの繰返しの終了後，必ず主ファイルからレコードを読み込むので，主ファイル間のマッチング処理はできない．このように MOTHER SYSTEM では出力を中心に考えたため，入力のタイミングが複雑

なプログラムを記述するのは難しい.

Pansophic Systems 社の EASYTRIEVE[9] では,レコードの選択,演算,転送条件を各入力ファイルごとに分離して記述することで,制御を簡潔に記述しようとしている.

Pro Computer Science 社の PRO-IV[8] では計算を行うタイミングを指定するために,データ構造にロジック番号を記入する.利用者は,指定したロジック番号ごとにそのとき行われる計算ロジックを記述すると,システムが生成した制御ロジックにこれらの計算ロジックが挿入されてプログラムが生成される.

MJS 社の JSP-COBOL[8] はジャクソンの図式表現を用いて制御ロジックと個々の計算ロジックを別々に書いたものを,COBOL プログラムに変換するツールである.

5. 仕様コンパイラ

従来の手書きのプログラム設計書と同様の書式をもつ複数のディスプレイ画面に,プログラム仕様を『いかにではなく,何を』の形式で指定し,これからプログラミング方式あるいはコンパイラ方式でプログラムを生成する自動プログラミングシステムがある.このようなシステムでは,仕様要素が従来の高級プログラミング言語の命令より抽象度が高く,記述も非手続き的であることが多く,また従来の設計書に近い形態で処理内容を記述できるので視覚性にも優れている.これらをここでは,設計仕様からプログラムへのコンパイラ,すなわち**仕様コンパイラ**と呼ぶ.

例としては,電力中研の原田らによって開発され東芝エンジニアリングによって実用化された SPACE[10),11)],日立の DSL[17] や HISPOT[28],富士通の BAGLES[43] などがある.

5.1 SPACE

SPACE を用いたプログラム仕様作成作業は,ワークステーションのディスプレイ上に表示された**図-10** に示すような8種の図/表形式の設計画面に,与えられた仕様要素ーあらかじめ標記法と意味が定められた図形や文で,設計パターンと呼ばれるーから適切なものを選んで記入するといった"はめ込み編集作業"である.したがって,従来の手書きの仕様書作成作業に比べると,目標や方法がはっきりしており効率的である.

たとえば図-10 は,給与マスタ更新ジョブの仕様を入力したときのいくつかの設計画面を示している.①は『このジョブが給与計算プロセス唯1つからなり,このプロセスは3つのファイルを入力し,1つのファイルと2つのレポートを出力する』ことを,②はこのジョブで使われるファイルの諸属性を,③は『給与計算プロセスが6つのモジュールからなる』ことを,④は『グループキーK2を給与マスタの部NOと社員NOがこの順で対になったキーである』ことなどを,⑤は『給与マスタがK1の値が一定の部の繰返しからなり,その中にK2の値が一定のレコードが繰り返している』ことを,⑥はこのジョブを構成するプロセスやモジュールの名前と説明を,⑦は『給与マスタの部NOから新給与マスタの部NOへのデータ移送を記号A1で表わす』ことなどを,おのおの指定している.

このようにして,ジョブのプロセス分割,プロセスのモジュール分割を指定した後,最後にモジュール機能をデシジョンテーブル・モデルに基づいた方法で,ロジックテーブルに処理と条件とその対応付けに分離して記述する.

SPACE の最大の特徴は,モジュールが **2.2** で分類した制御ロジックに従ってデータを処理する相異なる処理状態を,コンディションパターンという関数(たとえば,MC(ファイルF;キーK)はキーKでマッチング中のレコードがファイルFにあればY,なければN,Eof レコードならEを返す関数)の相違なる関数値として,簡潔に表現できるようにしたことである.たとえば,⑧は MC を用いて,給与更新モジュール(Kyuyo_Kousin)が,給与マスタと残業ファイルのK2キーにおけるマッチング処理を行うことを表している.具体的には,『このマッチング処理はA9(ACTION 部の9行目のこと)の do_again 文が示すように給与マスタと残業ファイルがともに Eof になる(規則R6)まで繰り返す.また両ファイルに同一キー値のレコードがあったとき(規則R1)は,A1 において残業計算モジュールを呼び出し,このキー値をもつ残業レコードに対してその読込みと残業の集計を行う.一方,給与明細のリスト出力については,給与マスタレコードがあるとき(規則R1,R2,R3)に,給与プリント出力モジュール(Kyuyo_Output)を毎回呼び出して1レコード分の出力処理を行う』ことなどを示している.

このように,制御ロジックについては対応するコンディションパターンを条件部に併置する方式で組み合わせるだけでよく,後は業務固有の計算ロジックを指定するだけでプログラム仕様を作成できる.さらにこ

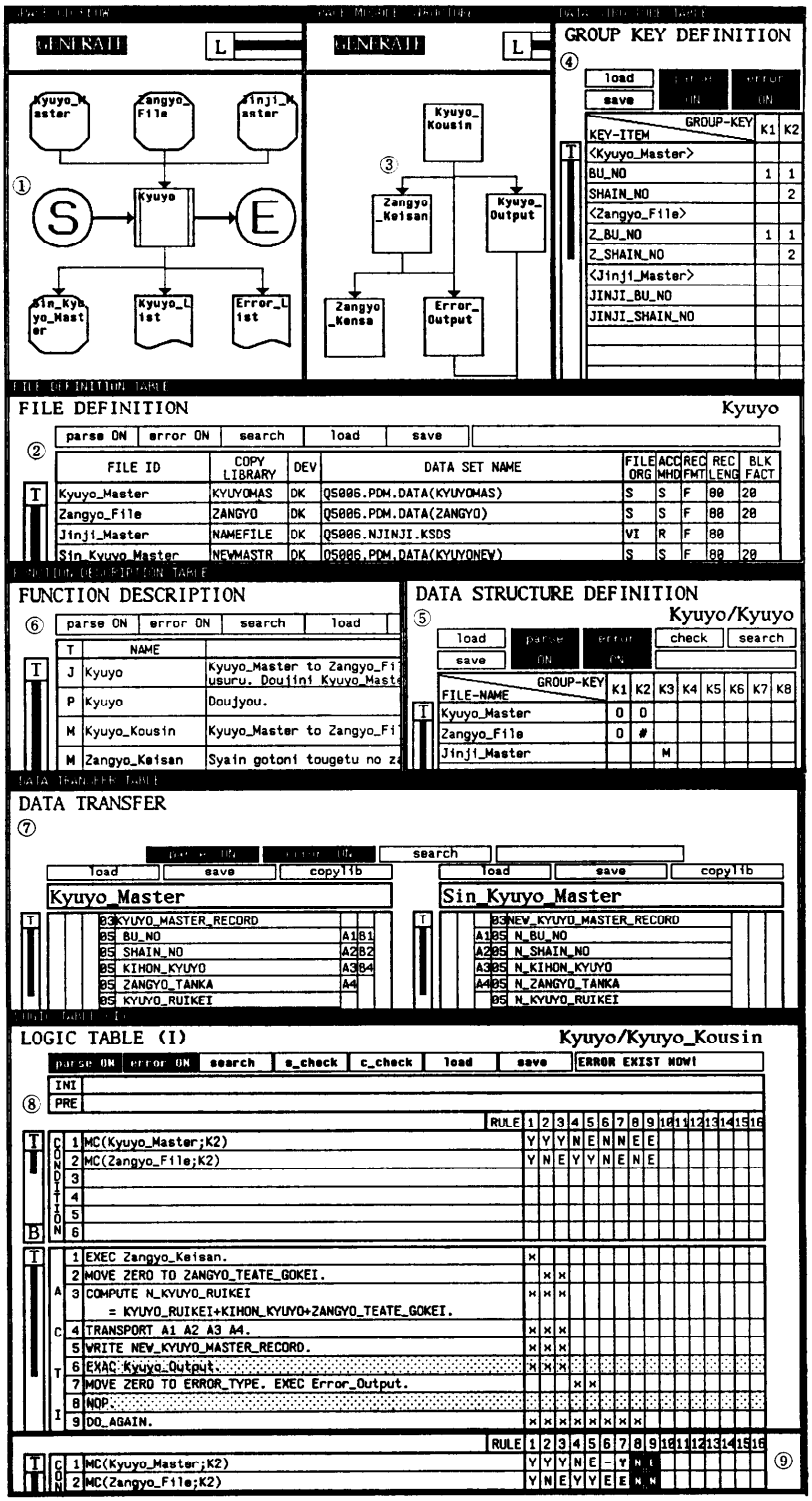


図-10 SPACE によるプログラム仕様記述例

の組合せは自由度が高く複数ファイルのマッチング処理や多段のグループ処理を容易に記述できる．このように SPACE は，部品の規模を小さくし，これらの組合せ指示を設計パターンとデシジョンテーブルを用いて自由にかつ分かりやすく表現できるようにしているので，適用性の高いシステムとなっている．

SPACE はコンパイラ方式でコードを生成する．具体的には，ロジックテーブルを解釈実行する基本的な制御構造に，各設計パターンごとにそれが表す定型処理を記述したプログラムコード群を，プログラム仕様に従って編集し埋め込むことによってプログラムを生成する．

このようなコード生成機能に加え，仕様獲得機能の一部として，仕様の表現方法の正しさや完全性（記述に漏れや矛盾がないか）などを検査する機能がある．たとえば，⑧では『A 6で EXEC を EXACと タイプミスしている，処理A 8に対する実行指示がない』などをアミ掛けで警告している．また⑨では規則R 1からR 6までを指定したときに C_CHECK メニュを指定すると『R 7が太字表示されこの規則が重複指定されている，R 8からR 9が反転表示

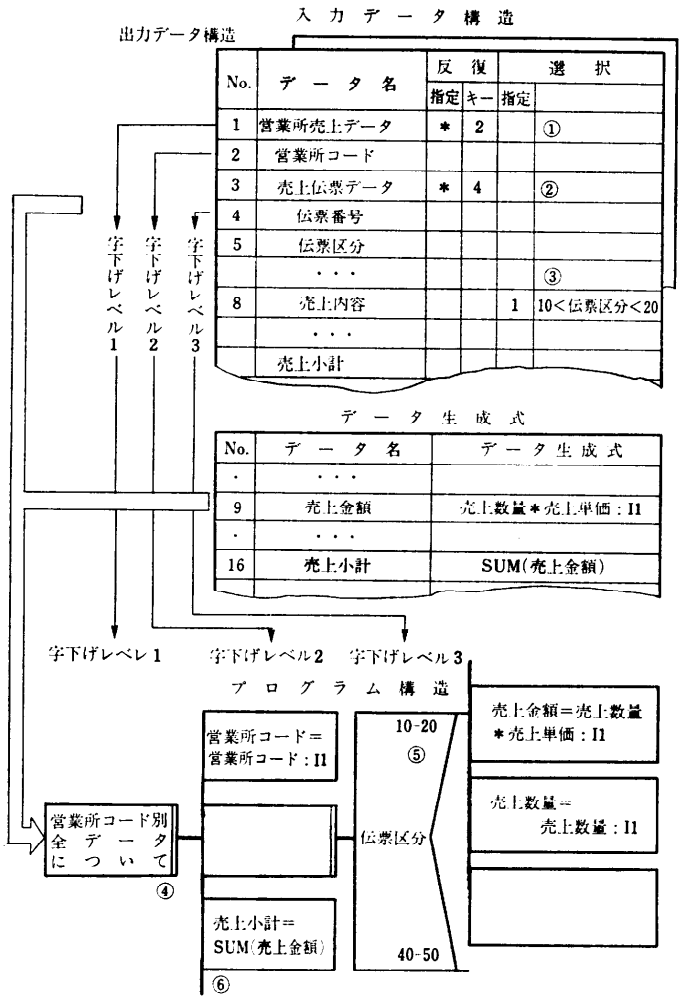


図-11　DSL によるプログラム仕様と生成プログラムとの対応

されこれらの規則が未指定である』ことを警告している．これによって設計の漏れや矛盾を早期に発見することができ，生成されたプログラムの信頼性が向上する．

SPACE のプログラム仕様は図表形式なので視覚性に優れまた設計パターンは抽象度が高いので，これだけを設計書として維持管理すればよく保守作業も大幅に軽減される．

5.2 DSL

DSL は入出力データ構造と出力データの求め方を定めたデータ生成式からプログラムを生成する．

図-11 に営業所別売上合計リストを求める DSL によるプログラム仕様の一部と，プログラム生成の考え方を示した．

①は営業所別売上合計リストが営業所コード一定の営業所売上データの繰返しからなること，②はそれがまた伝票番号一定の売上伝票データの繰返しを含むこと，③は売上内容は伝票区分が 10 と 20 の間のときのみ売上伝票データの一部となることを，おのおの指定している．これらの出力データ構造を基に，ジャクソン法によってプログラム構造を決定する．実際，④の繰返し処理は②に対応し，⑤の選択処理は③に対応する．また，⑥に示すように，売上小計は営業所売上データに 1 つなので④の繰返しの終り処理として生成される．ただし，SUM 関数は最終的には加算命令に展開される．なお複雑な条件によって求まるデータに対しては，直接 COBOL でオウンコーディングする．

もし入力データ構造との間に構造不一致が起きれば，中間ファイルの決定および入出力タイミングの決定を行うことによって対処する．ただし，入力ファイルが複数本あるマッチング処理のときは対応付けが難しくなる．

5.3 HISPOT

HISPOT はオンラインデータエントリ・プログラム開発専用の部品合成方式による自動プログラミングシステムで，一連の表形式の対話画面に，業務種類（販売か仕入れかなど），伝票のレイアウト，有効データの範囲や属性，リストの形式などを指定するだけでプログラム仕様を完成できる．これは対象業務が限定されているので，プログラム仕様が汎用仕様をチューニング（特化）するだけで指定できるからである．このようなイージオーダ的開発方法は，多くのコンピュータメーカでオフコンシステムを中心に広く行われている．

5.4 BAGLES

BAGLES は金融業務専用で，**図-12** に示すようにしてモジュールの機能を，デシジョンテーブルに似た条件表記法で与えるところに特徴がある．また，業務

用語を『解約という条件は，取り引きコードが 04 のことである』などと定義して使うことができ，仕様が理解しやすくなる．

6. 要求コンパイラ

自然語や図などで記述した事務処理要求を入力し，これをデータディクショナリやデータモデルと突き合わせて理解し，固有のモデルに従った内部的な形式仕様に変換し，これを基に必要があれば中間ファイルを設計し，さらに求める結果を得られるような最適なプロセスフローを設計し，各プロセスをモジュールに分割し，最終的に各モジュールに対応するプログラムコードを作成し，これらを統合してプログラムを生成するような，一貫した自動プログラミングシステムを**要求コンパイラ**と呼ぶことにする．

残念ながら，このような機能をすべてもち，ある程度の領域に自由に適用できる実用的なシステムはまだ存在しない．しかし実験的システムとしては，MIT の G. Ruth, W. Martin, R. Baron, M. Morgenstern らによる Protosystem I[35]，富士通研究所の杉山らによる KIPS[38]，電力中研の原田，篠原らによる ARIES/I[9] などがある．

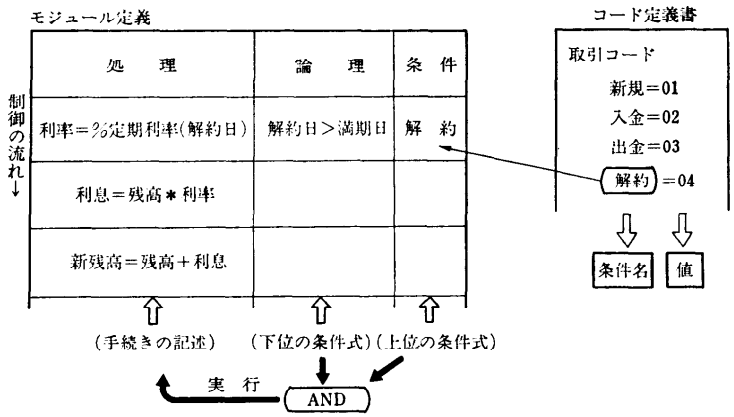


図-12 BAGLES によるプログラム仕様の一部[43]

6.1 Protosystem I

Protosystem I は上で述べた要求コンパイラのすべての機能を盛り込もうと計画しているが，現時点では SSL という形式言語で書かれた非手続き的な要求仕様からプログラムを生成するフェーズが完成している．

```
DATA DIVISION
(a) FILE SHIPMENTS-RECEIVED
        KEY IS ITEM
        GENERATED EVERY DAY
(b) FILE BEGINNING-INVENTORY
        KEY IS ITEM
        GENERATED EVERY DAY
(c) FILE TOTAL-ITEM-ORDERS
        KEY IS ITEM
        GENERATED EVERY DAY
(d) FILE QUANTITY-SHIPPED-TO-STORE
        KEYS ARE ITEM, STORE
        GENERATED EVERY DAY
(e) FILE QUANTITY-ORDERED-BY-STORE
        KEYS ARE ITEM, STORE
        GENERATED EVERY DAY
(f) FILE TOTAL-SHIPPED
        KEY IS ITEM
        GENERATED EVERY DAY
(g) FILE FINAL-INVENTORY
        KEY IS ITEM
        GENERATED EVERY DAY
(h) FILE REORDER-AMOUNT
        KEY IS ITEM
        GENERATED EVERY DAY
COMPUTATION DIVISION
① BEGINNING-INVENTORY IS FINAL-INVENTORY (1 DAY AGO)+SHIPMENTS-RECEIVED
② TOTAL-ITEM-ORDERS• IS SUM OF QUANTITY-ORDERED-BY-STORE FOR EACH ITEM
③ QUANTITY-SHIPPED-TO-STORE IS
        QUANTITY-ORDERED-BY-STORE      IF BEGINNING-INVENTORY IS GREATER
                                       THAN TOTAL-ITEM-ORDERS
        QUANTITY-ORDERED-BY-STORE
        * (BEGINNING-INVENTORY / TOTAL-ITEM-ORDERS)
                                       IF BEGINNING INVENTORY IS NOT
                                       GREATER THAN TOTAL-ITEM-ORDERS
④ TOTAL-SHIPPED IS SUM OF QUANTITY-SHIPPED-TO-STORE FOR EACH ITEM
⑤ FINAL-INVENTORY IS BEGINNING-INVENTORY - TOTAL-SHIPPED
⑥ REORDER-AMOUNT IS 1000 IF FINAL-INVENTORY IS LESS THAN 100
```

図-13 Protosystem I によるプログラム仕様記述例[35]

このフェーズは，与えられた仕様から計算を行う順序とファイルのレイアウトを設計することに主眼をおいている．

たとえば，問屋 (supplier) よりの入庫，小売店 (store) からの受注・出庫，問屋への不足発注処理を行う在庫管理要求の SSL による記述例を**図-13** に示す．まず利用者は，(a)～(h)に示すように，問題をモデル化するときに必要と考えた (FILE 句で始まる) 項目 (厳密にはその項目とキーを含むレコード) を，それがどのようなキーによって識別されるか (KEY IS 句)，発生頻度 (GENERATED 句) はどうかなどとともに独立に指定する．

次にこれら項目間の関係を①～⑥に示すような計算式の集合として，これまた独立に記述する．たとえば，②は各製品ごとの受注計はその製品に対するすべての店からの受注数の総和に等しいことを指定している．③は各店への出庫数は，在庫数が受注計より大きいときは受注数を，小さいときは受注数に (在庫数／受注計) を掛けたものに等しいことを指定している．

Protsystem I では，各計算に必要なレコード (すなわち項目) の集合を駆動データ集合と呼ぶ．例に対する各計算の駆動データ集合は**図-14** に示すようなグラフになる．この初期状態からスタートして，駆動データ集合がどのようなレコードをどのような範囲に対して含んでいるかに着目しながら，主としてファイルアクセスを最小にするという指針に従って，最適なファイル設計や最適ジョブステップ設計を行う．すなわち，同じ計算に同じ順序で使われたりそれから求められたりするレコード群を併合しできるだけ多くの項目を含んだレコードからなるファイルを設計したり，できるだけ順ファイルを使うようにしたり，駆動データ集合が同じ計算式をまとめて，できるだけ大きなジョブステップができるように計算をグループ化したりする．

したがってこのフェーズではたとえば，(e)は②と③で入力され，しかも②では店ごとに集計され，その結果の(c)がまた③の入力となっているので，②と③はグループ化できないが，③と①はグループ化できる．一方，④は(d)を集計入力するが⑤と⑥はそれを入力しないので，④と⑤と⑥をグループ化できる．したがって，(f)，(g)，(h)は1ファイルにまとめられる．また(a)と(g)もまとめられるが，これは両者のデータの発生場所が異なるので合理的ではない…などの最適化が行われているのではないかと思われる．

このような最適化の結果を基に，最終的にコーディングフェーズで各ジョブステップごとに，駆動データの構造に従って，制御ロジックを求め，これに計算式を割り当てることでプログラムを生成する．

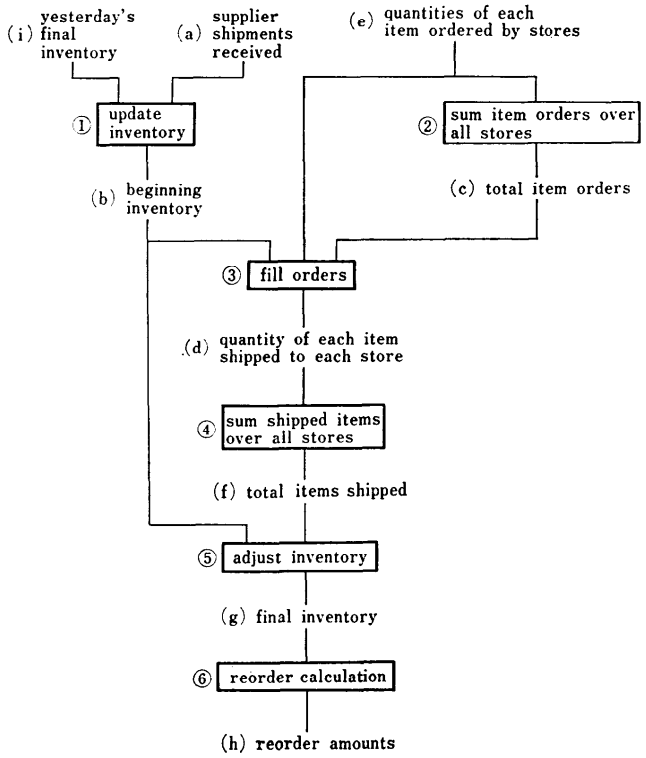


図-14 Protosystem I における最適化の初期状態[15]

6.2 KIPS

KIPS は仕様獲得に重点が置かれた要求コンパイラであり，要求の理解，プログラムモデルの構築，コード生成のすべての段階でその知識をフレーム形式のオブジェクトにもたせていることに特色がある．

図-15 に示すような日本語で記述された単文の集まりである事務処理要求を理解し，完全なプログラムモデルを構築する上での仕様上の問題点を分類し，それらを KIPS がいかに解決するかを以下に紹介する．

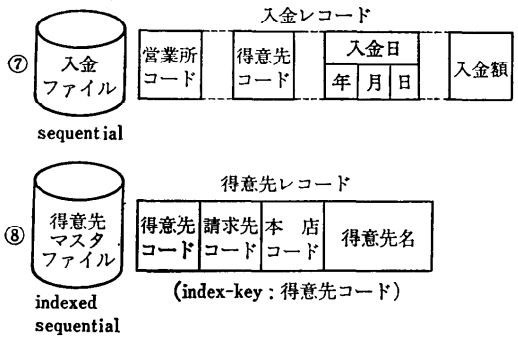
(1) 順不同，断片的 (①～③)

文①～③はチェック処理に関する入力

①，エラー出力先②，検査条件③を，順不同にかつ断片的に与えている．これらの関係は，**図-16** に示すように各文の構文意味解析結果をプログラムオブジェクトにメッセージとして送ることによって，下位のオブジェクトが能動的に未定義のスロットを埋めていくというサブゴール達成方式で明らかにされる．たとえば，文①を解析して，チェック処理を行うこと，その入力は #NYUUKIN-FILE（ディクショナリを引いて入金ファイルの名前をみつける）であることを知る．同様にして，文②からエラー出力スロットが #REPORT 1 になる．文③を解析し『～なら正常とする』に着目して，これがまだ未決定のチェック処理の正常出力ファイルの状態 #STATE 1 を定める条件を与えるものであると考え，この条件を一般的な条件を表すオブジェクト #COMPLEX-CONDITION に送ってそのインスタンスとして保持する．

① 入金ファイルをチェックする．
② エラーレコードはレポート出力する．
③ 入金額が 0 でなく，入金日の年が 56 以上，月が 1 から 12 の間で，日が 1 から 31 なら正常とする．
④ 得意先コードで得意先マスタファイルを検索し，得意先名，本店コードを付加する．
⑤ 入力はチェックの結果とする．
⑥ 付加した結果を得意先名でソートし，得意先名ごとに，入金額の平均，最大値，最小値を求める．

（a）要求仕様記述例



（b）ディクショナリより求めたレコード記述

図-15 KIPS が受け付ける要求仕様記述例[39]

（2） 不完全さ（レコード記述がない）

各ファイルの属性や⑦や⑧などのレコードレイアウトの定義などは，ディクショナリを参照したり，利用者に対話的に問い合わせることで補う．ただし，チェック処理のエラーファイルの #STATE2 は与えられなくても，デフォールトとして #STATE1 の否定と考える．

（3） あいまいさ（⑤の“入力”は？）

文⑤の入力としては，正常ファイルとエラーファイルがある，このうちどれを使うかは，利用者に問い合わせて解決する．

（4） 多様性（①⑤，④，⑥）

文①と文⑤では，処理の入力の指定法が異なる．文

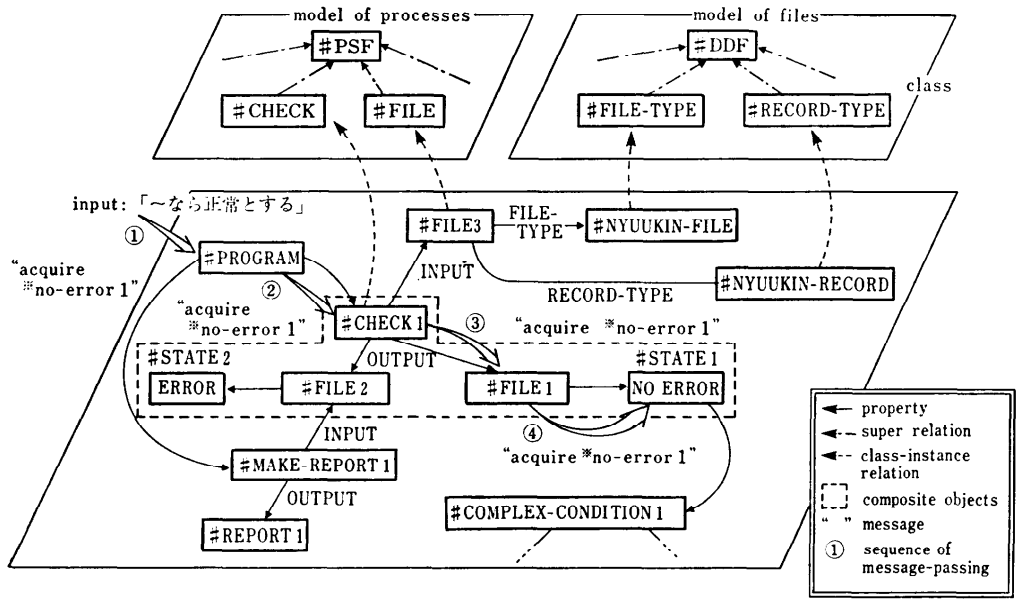


図-16 KIPS における仕様獲得とプログラムモデル構築[39]

①では，入力対象を文の目的語として表現しているのに対し，文⑤では，入力が何であるかを言明している．文④では複数の動詞で1つのオペレーション（項目追加処理）の各側面を規定している．現在 KIPS ではこの種の動詞として“検索”，“付加”，“出力”の3つを許している．一方，文⑥では，“ソートし”と“求める”という2つのオペレーションを，1つの文で表している．このように，動詞，オペレーション，文の対応がさまざまである．この対応関係の解析は構文意味解析あるいは文脈解析で行われる．

（5） 多義性（①の“出力”，⑤の“入力”）

たとえば，“入力”とか“出力”などの語は，入出力ファイルを指したり，オペレーションの指定の一部であったりする．これらの判別は構文意味解析で行われる．

（6） 文脈参照（⑤⑥の“結果”）

文脈解析では，レポート出力以外のオペレーションに関して，最近情報が追加されたオペレーションを優先することにしている．したがって，文⑤の“結果”は文①のチェック処理の出力であり，文⑥の“結果”は文④の付加処理の出力であると考える．

以上のような解析の後，KIPS は完成したプログラムモデル（プログラムオブジェクトのインスタンス）から HYPER COBOL プログラムを生成する．このとき，プログラムモデル内の各オブジェクトが HYPER COBOL のコードを1オペレーションに対して1ユニットの形式で出力する．

「テーマ台帳に登録されたテーマを担当する研究者に対して，氏名コード，氏名，所属する研究室名をレポートせよ．」

図-17　ARIES/I が受け付ける要求仕様記述例[9]

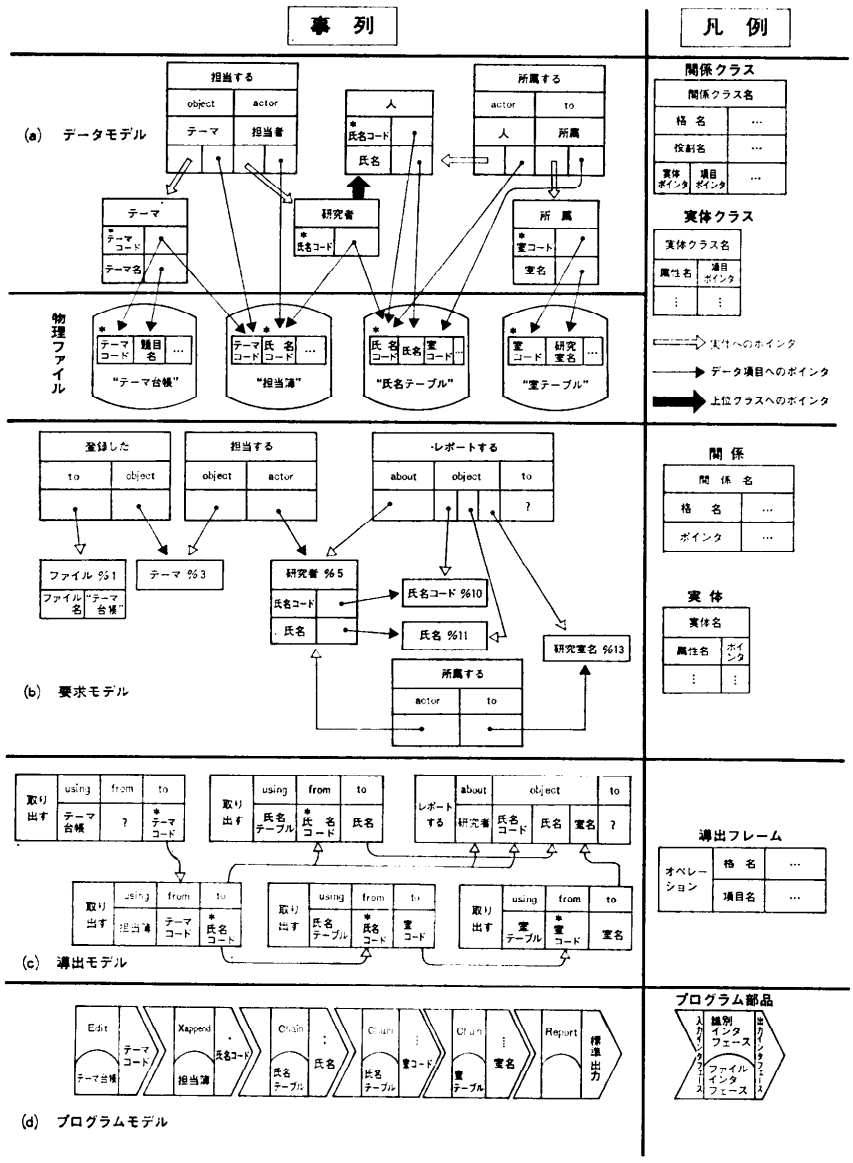


図-18　ARIES/I による仕様理解，設計，コーディング[9]

6.3 ARIES/I

ARIES/I は，図-17 に示すような単文からなる事務処理要求を受け付け，これを通常のプログラマが行うのと同じ方法で，理解し，設計し，コードを生成する各作業を，自動的に行う要求コンパイラである．

ARIES/I では，各要求は特定の言い回しを使って表現される．この言い回しは特定のオペレーションを表す動詞とそのオペランドを表す名詞からなるが，オペランドはさまざまな修飾によって限定できる．

要求を理解する段階で，ARIES/I は図-18(a) に示すようなデータモデルを用いる．これは企業活動を表すのに必要な実体およびその間の関係を表現したもので，この関係には実体間の上位下位関係と，あるファイルのレコードにともに含まれること(同居関係)によって生じる意味的な関係（たとえば，氏名テーブルは氏名コードをもつ人が室コードで表される室に所属することを表している）がある．要求仕様を構文解析した結果をデータモデルと照合することで，要求に現れる項目がどのファイルにどのような関係で入っているかが分かり，(b)に示す要求モデルができあがる．

この要求モデルはオペレーションを表す関係（たとえば，レポートする）とそのオペランドおよびそれを修飾している実体間の関係からなる．この実体間の修飾被修飾関係をデータの依存関係として位相ソートし，さらに各修飾関係がどのファイルにおける同居関係によって実現されているかを分析して，(c) の導出モデルを得る．この導出モデルは，要求文にあったすべてのオペランドがどのファイルのどのデータを基に求められるかを表している．たとえば，レポートすべき氏名コードはテーマ台帳にあるテーマコードと担当簿において同居関係にあることから求められる．

次に，導出モデルにおける各導出関係において，その関係を実現しているファイルの編成属性や，導出元(from 格) 項目が対応するファイルにおけるキー項目かどうかなどを基に，この関係を処理として実現するのに最適なプログラム部品を決定する．この結果，(d)に示すプログラム部品（とそれらをカスタマイズするパラメータ）の列として，プログラムモデルができあがる．なおこのとき，途中結果が次のファイルのキーの順に並んでいないときにはソート処理を追加したり，複数レコードに対して1つの項目値が定まるときにはグループ処理を追加する．また，グループ処理はできるだけ早い段階で行い，同一レコードの無駄な重複を避けるように最適化を行っている．

各プログラム部品は HYPER COBOL プログラムを生成する PROLOG プログラムであり，最終的にプログラムモデルにコード生成メッセージが送られると，HYPER COBOL プログラムを生成する．

このように，ARIES/I では設計知識などの汎用知識はモデルの変換規則として，要求理解やプログラミング用の知識は個々の部品知識としてもっている．

6.4 阪大のシステム

阪大の上原，藤井，豊田らは自然語で記述された仕様における目的をゴールとして，そのゴールからそれを達成するためのプランを，プランからよりプリミティブなプランを推論するといった段階的詳細化を行いながら PROLOG プログラムを生成する研究を行っている[41]．

7. その他ツール

1. にあげた自動プログラミングシステムの定義には当てはまらないが，プログラム開発作業の一部を自動化しているようなシステムや研究について簡単に紹介する．

7.1 データ定義部プロトタイピングツール

事務処理プログラムでは，ファイルレイアウトやオンライン画面や帳票などのデータ定義部の作成が他の分野に比べると量的に多く手間がかかる．このため，これらをディスプレイ上でそのプロトタイプを目で確認しながら対話的に設計し，対応するデータ定義部やそれに含まれるデータ項目の値検査モジュールを自動生成するデータ定義部プロトタイピングツールがある．たとえば，日本電気の SEA/I[23]，日立の EAGLE/II[5]，富士通の ADAMS[8]，東芝の MYSTAR[13]，IBM の CSP[8]，CANO-AID, ALICE, PRO-IV, STAMPS, SPCLAN などはこの機能を有する総合開発環境である．

7.2 部品合成支援ツール

プログラムの主制御を表す制御部品の名前と，この部品を問題に合わせて特化するためのファイル名やキー名などの結合パラメータを専用画面からメニュ方式で入力し，プログラムの制御ロジックを生成する機能を，SEA/I，EAGLE/II，CSP などはもっている．

7.3 日本語プログラミングツール

COBOL 命令と1対1に対応する日本語で記述された命令を用いて，読みやすいプログラムを作成し，これを COBOL に変換するツールとして，日立の漢字

CORAL[6] や長野県信連の木村らの事例[18] がある.

7.4 チャートエディタ

プログラム表現を視覚化するために,(主として木構造の)チャート記述法に従ってプログラムを図形と文章で入力し,これをプログラムに変換するツールとして,日本電気の SPDTOOL[19],日立の SDL/PAD[22],JBA の HCP チャートプロセッサ[37],富士通の YACⅡ[26],富士ソフトの COMPAL[8],東芝の IMAP[24],DEC の VAX COBOL GENERATOR[14] などがある.

7.5 要求仕様獲得システム

要求仕様の獲得を中心にした研究例を紹介する.

京大の大西,阿草,大野らは言い回しを限定した日本語(JRDL 表現という)で記述された要求仕様を,16 種の関係を中心とした CRDL 言語で記述した形式仕様へと変換し,これを基に要求中の矛盾や不完全性の発見と解決を行うシステムを開発している[32].

東京工大の佐伯,米崎,榎本(現富士通国際研)らは TELL というシステムを開発し,自然語で記述された仕様を述語論理で記述された形式仕様に変換する研究を行っている[36].

IPA の宮下らは,オブジェクト指向によるプロトタイプ実行機能,結果のアニメーションによる表示機能,ペトリネットによるタイミング解析機能などをもった要求仕様獲得支援システム SKETCH[25] を構築している.

7.6 設計支援ツール

プロセスフロー設計やファイル設計などを(半)自動化するツールを紹介する.

MBA 社の M. Bryce らによる PRIDE-ASDM[8] は,メニュ画面に入力した出力要素に関する情報から,データの依存関係,データの発生や利用に関するタイミングなどを基に,最適なプログラム分割やファイル設計を行う.

日立の ICAS-REUSE[4] は,先に述べた京大の大西らと同様の方法で,日本語で記述された要求文を格文法を基に解析し,システム機能とデータの関係図(データフロー図)へ変換する.このようにして得られたデータフロー図は多くの場合,入力と出力が完全に繋がっていない場合が多い.ICAS-REUSE は,既存の完全なデータフロー図群の中から,要求データフロー図と最も類似性の高い(共通部分が多い)データフロー図を選び,いかにして入力から出力が得られるかを与える処理フローを要求データフロー図中に補う.

このように ICAS-REUSE は既存の仕様書の再利用を自動化して設計を支援している.

南カリフォルニア大の Neighbors らによる DRACO[27] では,利用分野ごとにドメイン記述言語で要求仕様を記述し,システム分析者やシステム設計者がこれを他の記述言語に変換した結果を再利用することを支援している.

8. おわりに

紙数の都合ですべてのシステムを同様に詳しく説明することはできなかった.したがって,特徴的なもの代表的なものを中心に説明した.

事務処理プログラムは **2.** で述べたように他の分野のプログラムに比べてパターン化しやすく,また多くのプログラムモデルが研究されている.したがって,自動プログラミングの研究や実用化も他の分野に比べて盛んである.この傾向は∑計画と相まって国内ではよりいっそう盛んになると思われる.近いうちに COBOL や PL/I でプログラムを書く時代は幕を閉じるであろう.そのときは,仕様コンパイラなどがプログラム開発の中心となっているであろう.研究としては,要求コンパイラやプログラム理解によるプログラムからプログラム仕様への逆変換などが中心となると思われる.このときには,業務固有の概念の知識を基に制御ロジックのみならず計算ロジックの生成までも行えるであろう.多くの研究者の今後の努力に期待したい.

参 考 文 献

1) Barr, A. and Feigenbaum, E. A.: 人工知能ハンドブックⅡ, (田中幸吉, 淵 一博), 共立出版, p. 566 (1983).
2) Balzer, R.: A 15 Year Perspective on Automatic Programming, IEEE Trans. Softw. Eng., Vol. SE-11, No. 11, pp. 1257-1268 (1985).
3) Boehm, B. W.: Software Engineering Economics, Prentice-Hall, p. 767 (1981).
4) 千吉良英毅他:システム仕様書の再利用によるソフトウェアの開発技法 (ICAS-REUSE), 日立評論, Vol. 69, No. 3, pp. 249-154 (1987).
5) 葉木洋一他:システム開発支援ソフトウェア EAGLE, 日立評論, Vol. 68, No. 5, pp. 373-378 (1986).
6) 原田寛郎他:オンラインデータベースシステム向き業務プログラム開発維持支援システム:漢字 CORAL, 日立評論, Vol. 64, No. 5, pp. 39-42 (1982).
7) 原田 実:ソフトウェア生産性向上ツール(下)

ーオートメーション方式編，日経コンピュータ，No. 65, pp. 175-199 (1984. 3. 19).
8) 原田 実：ソフトウェア開発支援ツールの現状，日経データプロ・ソフト (Mar. 1986).
9) 原田 実，篠原靖志：部品合成によるプログラム自動生成システム ARIES/I，情報処理学会論文誌，Vol. 27, No. 4, pp. 417-424 (1986).
10) 原田 実：仕様コンパイラ SPACE，情報処理学会第 33 回 (61 年後期) 全国大会論文集，pp. 609-610 (Oct. 1986).
11) Harada, M.: A SPECIFICATION COMPILER FOR BUSINESS APPLICATION: SPACE, Proc. of IEEE COMPSAC'87, Tokyo (Oct, 1987).
12) 本村昭二：データ・ディクショナリを活用したシステム設計法，日経コンピュータ別冊，pp. 203-211 (1986. 9. 10).
13) 飯塚まとひ他：OA 用ソフトウェア開発支援システム MYSTAR，東芝レビュー，Vol. 41, No. 8 pp. 23-26 (1986).
14) 柿本岳文他：グラフィカルな要求表現からの日本語 COBOL プログラムの自動生成，情報処理学会第 34 回 (62 年前期) 全国大会論文集，pp. 109-1096 (Mar. 1987).
15) 片岡雅憲：自動化技術，ソフトウェアの生産技法(菅野文友編)，日科技連，pp. 271-310 (1987).
16) 加藤誠喜：事務データ処理標準化パッケージ STAMPS のプリセットロジック，情報処理学会第 20 回全国大会論文集，pp. 147-148 (1979).
17) 河野史男他：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，日立評論，Vol. 65, No. 7, pp. 53-58 (1983).
18) 木村俊一：日本語プログラム，人工知能学会第 1 回全国大会予稿集，pp. 465-468 (1987).
19) 岸本誠司他：SPD 支援ツール SPDTOOLS，構造エディタ (原田賢一編)，共立出版，pp. 121-134 (1987).
20) 古宮誠一：部品合成によるプログラム自動合成システム PAPS—知識工学アプローチを用いたその実現方式—，電信技報，SS 87-2, pp. 5-12 (1987).
21) 久世和資：ストリームを扱う言語 STELLA による在庫管理システムの記述，情報処理，Vol. 26, No. 5, pp. 497-505 (1985).
22) 前沢裕行他：プログラム自動生成システム SDL/PAD，日立評論，Vol. 66, No. 6, pp. 65-70 (1984).
23) 松本正雄他：ソフトウェア CASE ファシリティ：SEA/I，情報処理学会第 27 回全国大会論文集 4 B-1, pp. 513-514 (1983).
24) 松村一夫他：ソフトウェア設計記述法，東芝レビュー，Vol. 41, No. 8, pp. 6-10 (1986).
25) 宮下洋一他：SKETCH システムの全体構想，技術センター第 5 回発表会論文集，情報処理振興事業協会，pp. 43-46 (Nov. 1986).
26) 村上憲稔：YACII エディタ，構造エディタ (原田賢一編)，共立出版，pp. 135-146 (1987).
27) Neighbors, J. M.: The Draco Approach to Constructing Software from Reusable Components, IEEE Trans. Softw. Eng., Vol. SE-10, No. 5, pp. 564-573 (1984).
28) 内藤一郎他：オンライン対話型システムを対象としたプログラム部品再利用によるシステム開発法式とデータエントリシステムへの応用，システム工学会誌，Vol. 8, No. 2, pp. 13-24 (1984).
29) 西野甲矢三他：ソフトウェアの部品組み立て方式— MASCOT について—，情報処理学会第 33 回全国大会論文集，1F-4, pp. 587-588 (Oct. 1986).
30) 西村高志：MOTHER SYSTEM によるソフトウェアの設計と製造，情報処理，Vol. 25, No. 11, pp. 1247-1254 (1984).
31) 大木幹雄：部品再利用支援ツール PRECOBOL, Software Tools Symposium '87 予稿集，pp. 217-230, 情報サービス産業協会 (Jan. 1987).
32) 大西 淳他：要求定義のための要求フレーム，情報処理学会論文誌，Vol. 28, No. 4, pp. 367-375 (1987).
33) Prywes, N. S. and Pnueli, A.: Compilation of Nonprocedural Specifications into Computer Programs, IEEE Trans. Softw. Eng., Vol. SE-9, No. 3, pp. 267-279 (1983).
34) Rice, J. G. (山口圭一他訳)：自動プログラム構築技法の実際，近代科学社，p. 339 (1983).
35) Ruth, G.: Protosystem I: An Automatic Programming System Prototype, Proc. of the NCC, Anaheim, Calif., AFIPS 47, pp. 675-681 (1978).
36) 佐伯元司他：自然言語の語い分割による形式的仕様記述，情報処理学会論文誌，Vol. 25, No. 2, pp. 204-215 (1984).
37) 佐藤 学：JBA の HD システム，事務管理，Vol. 25, No. 11, pp. 154-155 (1986).
38) 杉山健司他：対話型自然言語プログラミングシステムの試作，電子通信学会論文誌，Vol. J67-D (3), pp. 297-304 (1984).
39) 田中準一：アプリケーション開発支援システム：プログラム生成，情報処理学会第 32 回 (61 年前期) 全国大会論文集，pp. 579-580 (Mar. 1986).
40) 寺田賢二：開発支援システム JASMAC，ソフトウェア流通，No. 26, pp. 96-100 (1986).
41) 上原邦昭他：自然言語による仕様からの自動プログラム合成，コンピュータソフトウェア，Vol. 3, No. 4, pp. 55-64 (1986).
42) Warnier, J. D. and Flanagan B. M.: Entrainement a la Programmation, Les Edition d'Organisation, Paris (1971).
43) 横山武史他：論理設計言語 BAGLES における部品化技術，プログラム設計技法の実用化と発展シンポジウム，情報処理学会，pp. 77-86 (Apr. 1984).
44) 油谷 泉：住商コンピュータの ALICE，事務管理，Vol. 25, No. 11, pp. 134-135 (1985).

(昭和 62 年 8 月 19 日受付)