ソフトウェア工学 100－6
（1994．9．8）

ファイル事務処理のモジュール設計の自動化システム－ＥＯＳ／Ｍ－
A module design formalism and its automation system EOS/M for file business processing problems

原田　実*，西村淳一**，田口志郎***
Minoru Harada,Junichi Nishimura,Shiro Taguchi

梗概

我々は過去に解アルゴリズムの作成とコーディングを自動的に行うシステムSPACEを開発したが，SPACEを使用するには，プロセスをモジュールに分割し(構造設計)，モジュールの機能仕様を条件式と実行文を用いて記述する(論理設計)など，いわゆる設計作業を行わなくてはならない．本論文では，この設計を自動化するシステムEOS/M(Equation Oriented Specification compiler for module design)について報告する．

EOS/Mはバッチ型のファイル処理問題を自動設計できる．EOS/Mの入力は，ファイル処理問題における実体や関連の属性項目間の関係と出力・更新要求を左辺が単一項目からなる等関係式という数式の集合で表した非手続き的仕様である．この非手続き仕様を表現するにはEOS仕様記述言語を用いる．EOS仕様では，等関係式中のデータ項目に対しては，その項目がどの実体の属性かを指定する識別子とその項目がどのファイルに存在するのかを指定するファイル修飾子を添える．

EOS/Mは，等関係式をその左辺に与えられる項目の値を求める計算式と考え，計算に必要なレコード集合が同じ計算式をグループ化し，モジュールを作る．モジュール内では項目間の導出関係分析から，計算式の実行順序を決定する．さらに，モジュールに割り当てられた計算式内の項目の識別子やファイル修飾子の特性からモジュールの制御ロジックを決定し，計算式の特性からその実行条件を決定する．これらの設計結果をSPACEの条件式や処理文に変換して，最終的に正確なSPACE仕様を生成する．

ABSTRACT

Almost all of the previous automatic programing systems, such as our SPACE, have set importance to the conversion from a formal specification to a program, namely to the creation of a solution algorithm and its cording in a programming language, but there are very few systems which automate module decomposition (structure design) and function specification (logic design). In this paper, we reports a system EOS/M (Equation Oriented Specification compiler for module design) that achieves automation of both design activities.

EOS/M can design batch type file processings. The input of EOS/M is a nonprocedural specification presented by a set of equations, each of which has only one item in its left hand side and expresses the relation between attribute items of entities and their relationships. In EOS specification, every item is affixed with an entity or relationship identifier and a file description.

EOS/M considers each equation as the computation to give the valueof its left hand side item, and collects such computations that require the same set of file records. This collection of computations becomes a module. Analysis of the derivation relationships among the items in the computations fixes the order of execution of the computaions belonging to each module. Control logics of the module are decided on the basis of the characteristics of the identifiers of the items in the computations. The timing when each computation is to be executed is determined based on its HBT type and file description. These design results are finaly expressed as the action statements and condition expressions of SPACE.

In a design experiment on several sorts of deta processing programs, EOS/M generates the complete SPACE module specifications of high quality at the same level as the human does.

*青山学院大学 理工学部 経営工学科

Department of Industrial Engineering, Faculty of Science and Engineering, Aoyama Gakuin University

**青山学院大学 理工学研究科 経営工学専攻

Department of Industrial Engineering, Graduate School of Science and Engineering, Aoyama Gakuin University

***青山学院大学 理工学部 経営工学科４年

Department of Industrial Engineering, Undergraduate School of Science and Engineering, Aoyama Gakuin University

1. はじめに

これまでの自動プログラミングの研究は，いわゆる(狭義の)プログラミングの自動化を対象にしていた．すなわち，そこでは，論理型[11]，オブジェクト指向型[2]，代数型[4]，数式[1,12]，決定表[8]，自然語[5,14]などの様々な仕様モデルによる形式的仕様から，解のデータや関係を導出するアルゴリズムを構築し，これを具体的な計算式の列として生成していた．ところが実際のソフトウェア開発(広義にはこれ全体をプログラミングともいう)では，このプログラミング段階以前に，構造設計とか論理設計という設計段階が必要である．

構造設計は，与えられた仕様を満足するソフトウェアを一連のプロセスという処理単位とその間を流れるデータに分解したり，各プロセスをモジュールという機能単位に階層的に分解することである．この分解は，個々のモジュールの解アルゴリズムが容易に得られるまで続けられる．一方，論理設計は，個々のモジュールに対して，それに要求されている機能を実現する論理すなわち解アルゴリズムを設計し，形式的に記述することである．

構造設計や論理設計の入力は，宣言的あるいは非手続き的に与えられた計算要件である．対象を事務処理ソフトウェアに絞ると，この計算要件を手続き的なプログラムに変換する研究として[6]，PrywesらのMODEL[12]，RuthらのProtosystem I [13]，河野らのDSL[10]，筆者らのARIES[5]，杉山らのKIPS[14]，などがある．しかし，MODELでは実際に要求されている計算に必要でない制御構造までも導出されてしまい最適化が難しい，Protosystem Iでは，複雑なデータ構造の扱いに対する考慮が欠けている，DSLでは複数本ファイルに対する照合処理が難しい，ARIESやKIPSでは日本文を入力仕様とするので複雑な計算条件を表現することが難しい…など様々な問題を含んでいる．また，この様な個々の問題以外に，これらの研究が扱う問題は実問題に比べ規模や複雑性において小さく，さらに構造設計を行っていない．

本研究の第1の目的は，対象をバッチ型のファイル処理問題に限定し，ファイルおよびファイル処理を定式化し，この定式化に基づいた等関係式仕様という仕様記述言語を提案することである．第2の目的は，等関係式仕様で記述されたジョブ要求を満足する一連のプロセスを設計し，次に各プロセスの仕様を満足する階層的に構造化されたモジュール群を設計する理論を確立することである．第3の目的は，プロセス設計の自動化システムEOS/P[9](Equation Oriented Specification compiler for Process Design)とモジュール設計の自動化システムEOS/M (Equation Oriented Specification compiler for Module Design)を開発し，我々の理論の有効性を実証することである．

EOS/Pについては次回の研究会で報告することにして，本論文ではEOS/M(Equation Oriented Specification for module design)について報告する．EOS/Mは，プログラムのモジュールへの構造設計と論理設計を自動的に行い，形式的なプログラム仕様を生成するシステムである．対象は事務処理の中心であるファイル処理に関する実規模程度の問題である．具体的には，左辺が単一の項目からなる等関係式という数式の集合として与えられた非手続き的な要求仕様から，構造化されたSPACEによるモジュール仕様群(これからCOBOLプログラムへの変換はSPACE[7]が自動的に行なう)を自動生成する．EOS/Mにおける自動設計の基本的手法についてはすでに報告しているので[8]，本論文においては非手続き仕様を記述する言語の詳細，システム化にあたって具体化された変換手法の詳細，多段階の照合処理を1プロセスに含めるために行った改良，および変換事例について論じる．

EOS仕様を構成する等関係式は，実体や関連の属性項目間の関係を算術式や条件式を用いて表現している．より正確には，左辺の項目を右辺の式で定義している．EOS仕様の各等関係式中のデータ項目に対して，それがどの実体あるいは関係の属性を表わすかを指定する識別子と，どのファイルにある項目なのかを指定するファイル修飾子を添える．この識別子とファイル修飾子によって，等関係式中に与えられる各項目の値を求める基となるレコード集合を正確に決定でき，この集合が同じ計算式毎にモジュール化する．さらに，計算式の種別やデータ項目間の導出関係分析からモジュール内での計算式の実行条件を決定する．これらの設計結果をSPACEの条件式や処理文に変換して，最終的に正確なSPACE仕様を生成する．

以下では，2節においてファイルとファイル処理を形式的に定義し，さらに生成されたプログラムが中間ファイルを用いずに処理できる条件を明らかにする．3節においては等関係式仕様記述言語を定義する．4節においては，設計結果を記述するSPACEのモジュール仕様を説明する．5節においては一段修飾の等関係式で表されたプロセス仕様からSPACE用のモジュール仕様群を生成する分解規則を明らかにし，EOS/Mの有効性を事例を用いて説明する．6節は結論である．

2. ファイル処理の定式化

事務処理では必ずその前提として，用いられるデータがどの様にファイルに格納されているかがあらかじめ定められている．これを図1の様に表現したものをデータモデルと呼ぶ．この例では，実体"社員"は実体ファイル"給与マスタ"で表され，実体"残業"は実体ファイル"残業ファイル"で表されている．また，"給与マスタ"は実体"社員"が実体"部"に"所属する"という関連をも表しているので関連ファイルとも考えられる．関連ファイルとは，例えば関連"所属する"の場合，部に所属する社員がn人いる時，これらの実体の識別子である部Noと社員Noを持つレコードを含むことによりこの1:n関係を表すファイルである．

ここではこのようなファイル処理を定式化すると伴に，自動設計の過程を説明するのに必要な諸概念を定義する．なお，EOSが扱うファイルとしては順編成ファイルのみを対象とする．索引編成ファイルやデータベースなどのより高次のアクセス法を持つファイルにはファイルの順次性という制約がないために，以下の議論は容易に拡張できる．

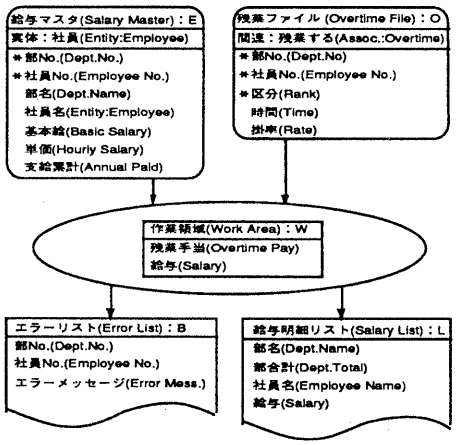

図1 給与計算問題に対するデータモデル
Fig. 1 Data model for the sample salary calculation problem.

2．1 ファイル，レコード，グループ

ファイルには，実体関連モデル[1]で言うところのいわゆる実体や関連の属性情報が格納されているとする．ファイル処理とは，これらのファイルから特定の実体[†1]の属性値を求めて，帳票を出力したりファイルを更新することである．一般に，xiを特定のドメインDiの要素として[†2]，レコードfは値xiの組<x1,x2,…,xn>[†3]であり，属性項目Xiはレ

[†1]以降では，紙数の都合で実体の属性を求める場合のみを議論する．しかし，この議論は識別子が複数項目からなるように変更するだけで容易に関連の属性を求める場合に拡張できる．

[†2]本節においては，英小文字は要素，英大文字は集合（リスト）を表す．また同一の文字の英小a／大A文字間にはa∈Aの関係があるとする．

コードfからドメインDiへの写像Xiはf↦Diであり，ファイルFは同一の型のレコードfのリスト{f}[†4]である．ファイルFの各レコードfがどの実体を表しているかを識別できる項目Iを実体の識別子と呼ぶ．またどの関連を表しているかを識別できる項目の組<I1,...,In>を関連の識別子と呼ぶ．この時，実体Iあるいは関連<I1,...,In>と呼ぶ．また，レコードを物理的に識別する仮想の識別子を考え，@で表わす．

事例として表1に示した部社員ファイルは，内容的には社員を表す実体ファイルであるが，部NOを含むので誰がどの部に属するかを表す関連ファイルとも考えることができる．なお，社員103のように複数の部に属するものもあるので，この関連は多対多である．また個々の関連を識別するのは<部NO,社員NO>と@である．

表1　事例としての部社員ファイルM
Table 1 A sample Department-Employee File.

部NO	社員NO	社員名	基本給	単価	@
A	103	山本一郎	290000	2500	001
A	105	川端二郎	280000	2300	002
B	106	石坂三郎	270000	2300	003
C	120	赤坂四郎	290000	2500	004
C	103	山本一郎	290000	2500	005

項目Dがファイル Fに属する項目であることをEOSではD_Fと表すが，論文中では字数を詰めるために添え字を用いてD_Fと表す．この時，添え字Fを項目Dのファイル修飾子と呼ぶ．この記法を用いて，ファイルFが識別子I2で識別子I1より先にソートされているなら，$I1_F \leqq I2_F$とする．また一般に，任意の識別子I_F,I'_Fに対して$@_F \leqq I_F$,$<I_F,I'_F> \leqq I_F$,$<I_F,I'_F> \leqq I'_F$である．表1では，$@_M \leqq$<部NO_M,社員NO_M>$\leqq$社員$NO_M \leqq$部NO_Mである．

ファイルFにおいて項目Xの値が等しいという同値関係によって分割された個々のレコードリストをグループと呼び，特にX(f)=xのものをgroup_F(X,x)と書く．

group_F(X,x)={f|∃f∈F,X(f)=x}[†5]…(1)

2．2　識別子による属性の修飾

ファイルFが実体を表し，その識別子がI_Fで，項目X_Fが実体I_Fの属性である時，実体I_Fの属性X_Fの値リストを$X_F.I_F$と書く．この時，項目X_Fは識別子I_Fで修飾されているという．しかし，1つの実体の属性が複数のファイルに存在する時や，新しいファイルを作成する時には，属性のファイル修飾子とそれを修飾する識別子のファイル修飾子とが異なることもある．従って，一般的には，$X_F.I_G$はファイルG内の各レコードgの識別子Iの各値iに対するファイルF内のグループgroup_F(I,i)内の各レコードfの項目Xの値リストの内容をiの昇順に並べたリストとして定義される．この様な"."1つによる，実体Iと属性Xの修飾を一段修飾という．

$X_F.I_G$={X(f)|∃g∈G,∃f∈group_F(I,I(g))}…(2)

例えば，部NO_M.社員NO_M={A,C,A,B,C}となる．明らかに(2)式は式(1)より，次のようにも書くことができる．

$X_F.I_{G=}$={X(f)|∃g∈G,∃f∈F,I(f)=I(g)}…(3)

この記法をさらに拡張して，ファイルFとGが実体Iの情報を含んでいたとして，Gで特徴付けられる性質を持たない実体Iの属性X_Fの値リストを$X_F.I_G$と書く．具体的にはファイルG内のレコードが持つ識別子Iの値以外の値i'を持つファイルF内の各レコードfの項目Xの値をi'の昇順に並べたリストとして定義される．

$X_F.I_G$={X(f)|∀g∈G,∃f∈F,I(f)≠I(g)}…(4)

しかし，$X_F.I_G$={X(f)|∃g∈G,∀f∈F,I(f)≠I(g)}は意味がない．なぜならレコードfの量化記号が∀なので集合を形成できないからである．なお，これらの記法はGが関連を表すファイルの時も同様に拡張できる．この時は識別子は<I1,I2>など実体識別子のペアであり，X_F.<I1,I2>$_G$などと書く．

この"."修飾は関連ファイルを介在することによって多段にすることができる．すなわち，出力／更新の要求単位となる実体I1を表すファイルGと属性Xを持つ実体I0を表すファイルF間をいくつかの関連ファイルRによって結合することによって，属性Xが実体I1とどの様な従属関係にあるかを"."修飾を複数回用いて指定できる．例えば，実体I1とRで表される関連を持つ実体I0の属性は$X_F.I0_G.<I1,I0>_R.I1_S$と表される．形式的には，属性Xの値を求める元になるレコードリストは，実体I1の各値i1に対して識別子<I1,I0>を持つ関連ファイルR内のI1(r)=i1なるレコードrと実体ファイルG内のI0(g)=I0(r)なるレコードgを経由して関係しているグループgroup_F(I0,I0(g))で与える．すなわち，

$X_F.I0_G.<I1,I0>_R.I1_S$={X(f)|∃s∈S,∃r∈R,∃g∈G,∃f∈F,I1(r)=I1(s),I0(g)=I0(r),I0(f)=I0(g)}…(5)

この修飾の段数はさらに関連ファイルを経由して重ねることができる．また，経由する途中ファイルを実体ファイルにすることもできるし，修飾元Uや修飾先Gを実体から関連に拡張することも同様にできる．例えば，$X_F.I0_G.<I1,I0>_R.<I1_S,<I2,I1>_{IT}.I1_U$={X(f)|∃u∈U,∀t∈T,∃s∈S,∃r∈R,∃g∈G,∃f∈F,I1(t)≠I1(u),I1(s)=I1(u),I1(r)=I1(s),I0(g)=I0(r),I0(f)=I0(g)}となる．この多段修飾が実際にどの様なファイルを経由して行われるのかを調査することがいわゆる要求分析であり，その結果を元にこの修飾関係を満たす属性リストを求める処理をプロセスやモジュールに分割しその仕様を定めることがいわゆる設計である．

2．3　等関係式によるファイル処理の定式化

ファイル処理に対する本研究の根底となる考え方は，『更新／出力ファイルLあるいは作業領域Lにおいて，識別子I0で表される実体$I0_F$の属性$X_L.I0_F$の値は，他のファイルG0におけるこの実体$I0_{F0}$の別の属性$X0_{G0}.I0_{F0}$や，この実体$I0_{F1}$と関連Rを持つ他のファイルG1における実体$I1_{F1}$の属性$X1_{G1}.I1_{F1}.<I1',I0'>_R.I0_{F1}$や，ファイルF2が実体I2を表すが同時に実体I0との関連も表すファイルである時には実体I2の属性$X2_{F2}.<I0,I2>_{F2}$などに，関数gを適用させて求まる』ということである．

$X_L.I0_F=g(X0_{G0}.I0_{F0},X1_{G1}.I1_{F1}.<I1',I0'>_R.I0_F,X2_{F2}.<I0,I2>_{F2},...)$…(6)

なお，ここで同じ実体を表す識別子Iが，ファイルによって異なるかもしれないので$I1_{F1}$と$I1'_R$の様に表したが，以後は簡単化のため同一記号で表す．また，識別子が所属するファイルがFやF0のようにいろいろあるが，これは式(6)によってある左辺の$I0_F$値に対応する右辺値を計算する場合は，この値を$I0_{F0}$値に持つ実体の$X0_{Gi}$などを用いることを示す．式(6)のように，左辺が単一の項目で右辺に左辺値を求める算術式や条件式を用いた等式を等関係式とよぶ．本研究ではファイル処理を，『等関係式の集合（これを等関係式仕様と呼ぶ）を入出力データ間の満たされるべき関係とみなし，この関係を満足するデータを出力・更新する処理』と定義する．

2．4　1パス性

等関係式仕様{$Y_L.O_F$=f($X1_{F1}.I1_{F1}$,$X2_{F2}.I2_{F2}$,..),...}において，ファイルF1とF2がファイル修飾子としてある等関係式fの左辺あるいは右辺の引数を修飾していれば，ファイルF1とF2は式fで連絡していると言い，F1<f>F2と書く．F1<f>F2かつF2<g>F3ならF1<f,g>F3とし，ファイルF1とF3も式fとgで連絡していると言う．この様にして，等関係式仕様{f}によってそこに現れるファイル{F}が互いに連絡していれば，仕様{f}は閉じていると言う．

プログラムが入力ファイルからレコードを順に読み込み，あらかじめ利用者が定義した有限の作業領域を用いて，データを作業ファイルに書き出さずに処理し，結果を出力ファイルに順に書き出す時，このプログラムは1パスであると言う．1パスとして処理できない原因にはさまざまなものがある．ここでは，個々の等関係式やファイルのソー

[†3] <K1,K2>は項目K1とK2を繋げた複合項目を表す．

[†4] リストとは重複を許す順序集合を意味する．また，断らない限り，同値分割などによって得られるリストの順序は元のリストの順序を保存するとする．

[†5] この時，結果のリスト内の要素の順序は，リストの定義式内に表れたリスト(ここでは，F)内の順序を保存するようにする．

ト状態によって判断できる3つの原因について論じる．

第1の場合は，先に説明したように等関係式仕様が多段修飾を含む場合である．この時は，式(5)のような多段修飾を持つ属性値を求めるために入れ子になったリストを内部的に保持する必要があり，このリストの長さが不定であるために作業ファイルへデータを掃き出す必要がある．この状態を回避するには，「全ての等関係式に出てくる全ての項目引数が一段修飾である」ことを要求する［一段修飾条件］．

第2の場合は，ファイル内のレコードの識別子によるソート順位の問題である．仕様中の全ての等関係式の引数が次の様に全て一段修飾としよう．

$X_L.I0_F=g(X0_{G0}.I0_{F0},X1_{G1}.I1_{F1},X2_{F2}.<IO,I2>_{F2},...)\cdots(7)$

この時，入出力ファイルL,F,F0,F1,F2,G0,G1などが識別子I0,I1,I2などIで異なる順にソートされている場合は，同一の実体に関する情報を異なるファイルから順に取り出して処理し出力することができない．いわゆる順序不一致の場合である．これを回避するには，「全ての等関係式に出てくる全ての識別子で入出力ファイルが同順にソートされている(されて出力される)」ことを要求する[順序条件]．

第3の場合は，異なるファイル内のレコードが共通の識別子で修飾されているかどうかの問題である．一段修飾のEOS仕様に含まれる共通の識別子Iを持つ項目$X_F.I_G$,$X'_F.I_{G'}$,..のファイル修飾子F,F',...に(次節の照合ファイルとして)含まれる全てのファイル名f,g,h,...においてI以上の識別子からなるデータ構造が異なっていると，X,X',...を求めるための照合処理ができないし，これらのファイルにあるより上位の識別子J(>I)による計算の繰り返しの中にこの照合処理が入れ子状に含まれない．従って，途中結果を作業ファイルに書き出さないと処理できない．これはいわゆる構造不一致あるいは脈絡不一致といわれるものである．この状態を回避するために，「識別子Iを共有するファイルは識別子Iより≦順位で大きい識別子を全て同じ順位で共有しなければならない」ことを要求する[構造条件]．

EOS/Pの後を受けモジュール設計を行うEOS/Mに与える等関係式仕様は閉じた1パス仕様に限定する．すなわち，仕様が閉じており，一段修飾条件，順序条件，構造条件を満足しているものとする．等関係式仕様がこれらの条件を満足すれば，EOS/Mは組込まれた設計知識によりプロセスをモジュールに分割し，そのSPACE仕様を生成する．

2．5　照合ファイル

例えば図1の事例における残業を行なった社員とそうでない社員のように，ある実体の属性(例では社員の残業単価と残業時間など)が異なるファイルに記録され，「ファイルに当該レコードがあるかないか」という事実が，このレコードで表される実体の状態(例では，残業をしたかしなかったか)を表現することがよくある．この状況は照合ファイルを用いて表現できる．照合ファイルF1&F2[I]とは，2つのファイルF1とF2の各レコードを両ファイルに共通の識別子Iを照合キーにして，その値が同じレコード同志を連結した仮想的なレコードからなるファイルである．なお，1つの照合に参加するファイルの数は共通のキーを持てば3つ以上でもよい．

$F1\&F2[I]=\{f1\hat{}f2|\exists f2\in F2,\exists f1\in F1,I(f1)=I(f2)\}$ †6…(8)

また照合ファイルF1&!F2[I]やF1&F2[I]は，実体Iを表すレコードがファイルF1やF2だけにあって他方にないファイルである．

$F1\&!F2[I]=\{f1|\forall f2\in F2,\exists f1\in F1,I(f1)\neq I(f2)\}\cdots(9)$

これらの照合ファイルは，いわゆるファイル処理における複数ファイルの照合処理を意味する．また，関係データベースにおけるジョイン演算に相当する[2]．照合ファイルは，実体Iの情報が複数のファイルに格納されている際に，これらの情報を実体毎に取り出し等関係式に従った計算を行うために必要となる．

2．6　漸化性

等関係式(6)で与えられる関数関係を解決する実際の計算について考える．等関係式内の項目は式(2)に与えられるようにリストである．等関係式の右辺のインスタンス間の対応は，左辺の識別子Iの各値i0に対する左辺値を，いる．この事を明示化するためにgの値を決める実際の計算を行なう単一の値を引数にする関数をg'(X,…)と書く．一般に，この関数は1つの算術式で表現できない．しかし例えば，B=ΣAは初期処理B=0と本体処理B=B+Aで表現できる．このように，関数g'が式(10)のように初期値を与えるx0と，記憶域中の中間結果g'(X')と現在レコード内の値xiからg'(X)を求めるg1に分解できる時，関数gが漸化性を持つという．

$$g'(X,\cdots)=\begin{cases}x0 & X\text{が空リストの時}\\ g1(xi,g'(X',\cdots),\cdots) & X=[X':xi]\text{の時}\dagger7\end{cases}\quad\cdots\cdots(10)$$

漸化性を持つ関数には，値リスト内での最大:MAX，総和:SUM，先頭値:1STなどを求めるものがあり，事務処理要求内の計算のほとんどがこのような関数で表わすことができる．

3．　等関係式仕様

前節で論じた式(6)や式(7)のような関数形式の等関係式では具体的な計算の意味を表せないし，また条件指定もできない．従って実用化の目的から，算術式や出力文や条件文を含む図2に示すような構文を用いて等関係式仕様を具体的に表すことにする．なお，これらの各式が関数gを適切に選べば式(6)の形式を持つことは容易に確認でる．また，どの等関係式も全て漸化性を持つことも分る．また構文的には，EOS/Mが対象とする一段修飾のEOS仕様とはファイル修飾子が全て単一ファイル修飾子の場合である．

また，識別子のファイル修飾子において照合ファイルがある場合，一般にその照合キーを明記するが，その照合キーが識別子に等しい場合は省略できる．さらに，項目のファイル修飾子がその識別子のファイル修飾子に等しい時は，それを省略できる(X_F.I_F=X.I_F).

```
<等関係式仕様>::=+{ <一般等関係式> }.
<一般等関係式>::=<等関係式>';'
 ¦IF '(' <基本条件式> ')' THEN (<等関係式>';' ¦ '{'<等関係式群>'}')
  [ ELSE (<等関係式>';' ¦ '{'<等関係式群>'}')]
 ¦SWITCH '(' <式> ')' '{' <CASE並び> '}'
 ¦<ファイル修飾子> '=' <ファイル名>
 ¦<実体識別子> '<' <実体識別子> ';'.
<CASE並び>::=+{ <CASE文> }.
<CASE文>::=<CASE> <式> ':' ( <等関係式> ';' ¦ '{' <等関係式群> '}' ).
<等関係式>::=<一意名> '=' <式> ¦ WRITE '(' <レコード名> ')'
 ¦DELETE '(' <レコード名> ')' ¦ REWRITE '(' <レコード名> ') ¦ NONE.
<レコード名>::=<一意名>.
<等関係式群>::=+{ <等関係式> ';' }.
```

②基本算術式

```
<式>::=<基本算術式> ¦ SUM '(' <基本算術式> ')' ¦ FIRST '(' <基本算術式>
')'
 ¦ LAST '(' <基本算術式> ')' ¦ MAX '(' <基本算術式> ')'
 ¦ MIN '(' <基本算術式> ')' ¦ FOLLOW '(' <基本算術式> ')' ¦ <定数>.
<定数>::= ALL (<表意定数>¦<文字定数>)¦<数字定数>¦<文字定数>¦<字類
定数>¦<表意定数>.
<基本算術式>::=+{+{+{<一意名>¦<一意数字定数>¦'('<基本算術式>')'¦**
'¦'*'¦'/'}¦'+'¦'-'¦'^'}.
```

③基本条件式

```
<基本条件式>::=<単純条件式>¦<否定単純条件式>¦<組合せ条件式>¦<否
定組合せ条件式>¦'('<基本条件式>')'.
<単純条件式>::=<比較条件式>¦<字類条件式>¦<正負条件式>.
<比較条件式>::=<比較条件記述項><比較演算子><比較条件記述項>.
```

†6 f1^f2はレコードf1の次にf2を繋げたレコードである．

†7 [X':x]はリストX'に値xを後に繋げたリストを表す．

```
<比較条件記述項>::=<一意名>¦<指標名>¦<>.
<比較演算子>::='>'¦'>='¦'=>'¦'<'¦'<='¦'=<'¦'=='¦'!='.
<字類条件式>::=<一意名>'==' <字類定数>.
<字類定数>::= NUMERIC¦ALPHABETIC.
<否定単純条件式>::='!' <単純条件式>.
<組合せ条件式>::=+{<基本条件式>:'&'¦'¦'}.
<否定組合せ条件式>::='!'('<組合せ条件式>')'.
```

④基本構文要素

```
<一意名>::=<データ項目名>['_'<単一ファイルid>]['('#(1,3){<指標及び添字付け>:','}')'].'<識別子並び>.
<一意定数>::= 定数'.'<識別子並び>.
<指標及び添字付け>::=(<指標名>¦<指標付けのないデータ項目名>['_'<ファイル修飾子>])[('+'¦'-')<整数>]¦<整数>.
<識別子並び>::=+{<識別子>:'.'}
<識別子>::=<実体識別子>¦<関連識別子>
<実体識別子>::=<データ項目名>'_'<ファイル修飾子>.
<関連識別子>::='<'<データ項目名>','<データ項目名>'>"_'<ファイル修飾子>
<ファイル修飾子>::=<単一ファイル修飾子>¦#(2,-){('('<ファイル修飾子>')"&')['['<照合キー名>']'].
<単一ファイル修飾子>::=(['!']<英字>)¦#(2,-){['!']<ファイルid>¦&'}['['<照合キー名>']'].
```

⑤字句要素

```
<名前>::=(<日英字>[#(0,18){['-']¦<日英数字>}]<日英数字>])'_'<ファイル修飾子>
<ファイルid>::=#(2,2){<英数字>}
<数字定数>::=[('+'¦'-')] +{<数字>} ['.' +{<数字>}].
<整数>::=*{<数字>}<0でない数字>*{<数字>}.
<文字定数>::=[N(A¦C¦H¦K¦N¦X)]("'"¦'"') *{<クォートでない文字>¦("'"¦'"')}("'"¦'"').
<表意定数>::=<表意数値定数>¦SPACES¦SPACES¦HIGH-VALUE¦HIGH-VALUES¦LOW-VALUE¦LOW-VALUES¦QUOTE¦QUOTES.
<表意数値定数>::= ZERO¦ZEROS¦ZEROES.
```

図2　等関係式仕様の構文規則
Fig. 2 Syntax of EOS specification.

以降において設計過程を具体的に説明するために，図1に示したデータモデルにおいて，以下に示すような処理要件を持つファイル処理問題を考える．なお，EOSが扱う問題では，同一ファイル中のREDEFINEされた異なるレコード記述中では，同一項目は同じ位置に指定されているものとする．

［処理要件］

1）給与マスタEに登録されている社員毎に給与明細リストLの各行を出力する．この際，社員が所属する部毎に部合計も印字する．

2）給与は基本給と残業手当からなる．

3）残業手当は当月の残業代を社員ごとに集計したものである．

4）残業代は，残業記録ごとにその残業区分に対応する掛率を使って下記の式で求められる．

残業代＝単価×時間×掛率

5）残業ファイルOの各残業レコードに対して，同じ社員Noを持つ給与マスタレコードがなかったり，同じ部Noを持つ給与マスタレコードがなかったりする場合は，その都度”残業社員エラー”や”残業部エラー”などのエラーメッセージをエラーリストBに出力する．

この処理要件を等関係式仕様で表現すると図3のようになる．なお，ファイル修飾子は図1のデータモデルで：に続けて示されたものを用いる．また，実体や関連の識別子も図1でその前に＊印を持つ属性として表明されているとする．さらに，実体ファイルおよび関連ファイル間の同一キーを持つレコードの対応多重度をn:mの様に示した．この事例では例えば，①は残業した社員の残業手当は，この社員の給与マスタ(E)の単価と残業レコード毎の残業時間とその残業の区分に依存した掛率の積であることを示している．②は残業しない社員の残業手当は0であることを示している．なお，②ではファイル修飾子中の照合キーが識別子に等しい例として省略されている．また，③では作業領域中(W)の給与は給与マスタ(E)の社員Noごとに計算され，それは社員No_Eで表される社員の給与マスタ内の基本給と社員No_Eで表される社員の作業領域中(W)の残業手当の和であることを，⑦ではリスト(L)の部合計は部Noごとに同じ部Noを持つ社員の給与の和であることを，⑩ではリストの明細行は部名，社員名，給与からなることなどを示している．同様にして⑫から⑳は給与マスタに同じ社員Noを持つ給与マスタレコードがない残業レコード(@_!E&O[社員No])毎あるいは同じ部Noを持つ給与マスタレコードがない残業レコード毎(@_!E&O[部No])にエラー処理を記述している．

```
①残業手当_W.社員No_E&O[社員No] = SUM(単価_E.社員No_E * 時間_O.<社員No,区分,@>_O&E[社員No] * 掛率_O.<社員No,区分>_O&E[社員No]);
②残業手当_W.社員No_E&!O = 0.社員No_E&!O;
③給与_W.社員No_E = 基本給_E.社員No_E + 残業手当_W.社員No_E;
④社員名_L.社員No_E = 社員名_E.社員No_E;
⑤給与_L.社員No_E = 給与_W.社員No_E;
⑥部名_L.部No_E = 部名_E.部No_E;
⑦部合計_L.部No_E = SUM(給与_L.<部No,社員No>_E);
⑧WRITE(明細行_L.社員No_E);
⑨WRITE(合計行_L.部No_E);
⑩明細行_L.社員No_E = FOLLOW(部名_L.部No_E^社員名_L.社員No_E^給与_L.社員No_E);
⑪合計行_L.部No_E = FOLLOW(部名_L.部No_E^部合計_L.部No_E);
⑫エラーメッセージ1_B.@_!E&O[社員No] = "残業社員エラー".@_!E&O[社員No];
⑬エラーメッセージ2_B.@_!E&O[部No] = "残業部エラー".@_!E&O[部No];
⑭社員No_B.@_!E&O[社員No] = 社員No_O.@_!E&O[社員No];
⑮社員No_B.@_!E&O[部No] = 社員No_O.@_!E&O[部No];
⑯部No_B.@_!E&O[部No] = 部No_O.@_!E&O[部No];
⑰エラー行1_B.@_!E&O[社員No] = FOLLOW(社員No_B.@_!E&O[社員No]^エラーメッセージ1_B.@_!E&O[社員No]);
⑱エラー行2_B.@_!E&O[部No] = FOLLOW(部No_B.@_!E&O[部No]^社員No_B.@_!E&O[部No]^エラーメッセージ2_B.@_!E&O[部No]);
⑲WRITE(エラー行1_B.@_!E&O[社員No]);
⑳WRITE(エラー行2_B.@_!E&O[部No]);
```

図3　給与計算問題に対する等関係式仕様
Fig. 3 EOS specification for the example salary calculation problem

4．　SPACEのモジュール仕様

モジュール設計を行うEOS/MはEOS/Pが行うプロセス設計の結果を受けて，正確なモジュール仕様を生成する．EOS/Mはこのモジュール仕様をSPACE用のロジックテーブル群として生成する．これはEOS/Mが直接COBOLの様なプログラムを生成するのは負担が大きいので，アルゴリズム構築とコード生成をSPACEにまかせ，EOS/Mは純粋に計算のモジュール化と計算の実行条件決定に集中するためである．図3の等関係式仕様に対してEOS/Mがモジュール設計した結果は図4に示すようなロジックテーブルで表されるが，ここではこれを事例に用いてSPACEにおけるロジックテーブルによるモジュール仕様の記述方法について説明する[6,7]．

一般にSPACEでは，ファイル処理プログラムを構成するモジュールを照合，集計，検索，計算型の4つに分類し，これらの処理を行うモジュールの実行状態Sを条件式とその値の組合せ{条件式C1＝値v1j,…}

で表す．実際，ロジックテーブルでは，「どんな状態Skの時どの処理{Ail,…}を行うか」を，「状態Skを表す規則列RkのCi行に値vikを記入し，この列内の処理Aj行に実行指示記号Xを記入する」ことで表現する．SPACEの最大の特徴は，ロジックテーブルの条件欄に記入して，モジュールの実行状態を簡潔に表現するのに便利な条件式がいくつか提供されていることである．例えば，ファイルFと他のファイルのキーKとでの照合処理の状態を表す条件式MC(F;K)は，Fから入力したレコードがキーKで表される現在照合中の対象を表わす状態にあることを=Yで，そうでないことを=Nで表す．一方，ファイルFのキーKによるグループ処理は条件式CB(F;K)か条件式LC(F;K)を用いて表わす．CB(F;K)は，グループ処理を行うモジュールMが，他から呼ばれた初期状態か，あるいはファイルFから入力したレコードによってキーKが一定の新しいレコードグループに入った状態の時=H(ead)を，グループ中の各レコード(先頭や最後のレコードも同様に含んで)処理状態の時=B(ody)を，現在のグループを終ろうとする状態の時=T(ail)を値にとる．LC(F;K)は，ファイルFから入力したレコードがグループ中の先頭レコードの時=F(irst)を，最後のレコードの時=L(ast)を，中間レコードの時=I(ntermediate)を，グループがこのレコードのみを含むとき=U(nique)を値にとる．これらの条件式は全てEOFレコードを入力した時は=Eを値にとる．一方，分類処理を表す"式D={式1;式2;…}"なる選択式は，式の値が式iの値に等しい時=iを，どの式iの値とも等しくない時=0を値にとる．SPACEには，これら以外にも配列処理用など計9つの条件式がある．これらの条件式を条件部に併置し，これらの値の組合せでモジュールの実行状態を表現する．

以下では図4の給与計算問題に対する4つのモジュール仕様を例に取り，SPACEでのモジュール仕様の記述法を説明する．部照合モジュール(Mod7)では，給与マスタEと残業ファイルOの<部No>キーにおける照合処理を条件式としてMCを用いて行うことを表している．具体的には，「この照合処理はA9のdo_again文が示すように給与マスタEと残業ファイルOが共にEOFになる(規則EE)まで繰り返す．残業をした社員がある部に対しては，すなわち両ファイルに照合対象レコードがあった時(規則YY)は，A1において社員レベルの照合モジュール(Mod6)を呼び出す．一方，残業した社員が所属する部がない時は(規則NY,EY)，残業部エラー処理を行う．」ことなどを示している．社員照合モジュール(Mod6)では，給与マスタEと残業ファイルOの<部No,社員No>キーにおける照合処理を行うことを表している．具体的には，「この照合処理はA11のdo_again文が示すように給与マスタEと残業ファイルOが共にEOFになる(規則EE)まで繰り返す．社員が残業をした時，すなわち両ファイルに照合対象レコードがあった時(規則YY)は，A1において残業計算モジュール(Mod0)を呼び出し，この社員に対する残業レコードを読み込み残業手当の集計を行う．一方，残業をしなかった時(規則YN,YE)は，A2において，残業手当に0をいれる．給与マスタEに社員のレコードがある時(規則YY,YN,YE)には，給与明細リストのレコード作成(A3～A5)と出力(A7)や部合計の集計と合計行の出力を行う部合計モジュール(Mod3)の呼び出し(A6)を行う．また残業した社員の給与マスタレコードがない時(規則NY,EY)には，残業社員エラー処理(A8～A10)を行う．」ことなどを示している．また，部合計モジュールは給与マスタEの<部No>キーに関する集計処理を条件式としてLCを用いて表している．具体的には，「給与マスタE内の各社員レコード(F,I,L,U)に対しては部合計への加算(A3)を，それらが各部の先頭レコードである時(F,U)は部名の転記(A1)と部合計の初期化(A2)を，最後のレコードである時(L,U)は部合計の出力(A4)をそれぞれ行い，これらの処理が終わると親モジュールに戻る」ことなどを示している．また，残業計算モジュールは，残業ファイルOの<部No,社員No>キーに関する集計処理をCB式を使うことによって表わしている．実際，「残業ファイルOから残業レコードを繰り返し入力するごとに，それが<部No,社員No>キー値が同じ新しい集団に入った瞬間(H)なら残業手当を初期化し(A1)，同じ社員に対する残業レコード集団内の各レコード処理状態である(B)なら単価と時間と掛率を掛けた値を残業手当に加え(A2)，レコード集団の終わり(T,E)なら集計値を伴って親モジュールに戻る」ことなどを示している．

Mod7　CondExp = MC　Module->FileModifier EO

C1	MC (給与マスタE;<部No>)	Y	Y	Y	N	N	N	E	E	E
C3	MC (残業ファイルO;<部No>)	Y	N	E	Y	N	E	Y	N	E
A1	exec　Mod6.	X	X	X						
A2	エラーメッセージ2 OF エラーリスト := "残業部エラー".				X	X				
A3	部No OF エラーリスト = 部No OF 残業ファイルO.				X	X				
A4	社員No OF エラーリスト = 社員No OF 残業ファイルO.				X	X				
A5	XWRITE エラー行2 OF エラーリスト.				X	X				
A6	do_again.				X	X	X	X	X	

Mod6　CondExp = MC　Module->FileModifier EO

P1	IF DT07--MC--<部No> NOT = 給与マスタE--<部No>--C THEN EXIT.									
P2	IF DT07--MC--<部No> NOT = 残業ファイルO--<部No>--C THEN EXIT.									
C1	MC (給与マスタE;<部No,社員No>)	Y	Y	Y	N	N	N	E	E	E
C3	MC (残業ファイルO;<部No,社員No>)	Y	N	E	Y	N	E	Y	N	E
A1	exec　Mod0.	X								
A2	残業手当 OF 作業領域 := 0.		X	X						
A3	給与 OF 作業領域 := 基本給 OF 給与マスタE + 残業手当 OF 作業領域.	X	X	X						
A4	給与 OF 給与明細リスト := 給与 OF 作業領域.	X	X	X						
A5	社員名 OF 給与明細リスト := 社員名 OF 給与マスタE.	X	X	X						
A6	do　Mod3.	X	X	X						
A7	XWRITE 明細行 OF 給与明細リスト.	X	X	X						
A8	エラーメッセージ2 OF エラーリスト := "残業部エラー".				X		X			
A9	社員No OF エラーリスト = 社員No OF 残業ファイルO.				X		X			
A10	XWRITE エラー行2 OF エラーリスト.				X		X			
A11	do_again.	X	X	X	X	X	X	X	X	

Mod3　CondExp = LC　Module->FileModifier E

C1	LC (給与マスタE;<部No>)	F	I	L	U	E
A1	部名 OF 給与明細リスト := 部名 OF 給与マスタE.	X			X	
A2	部合計 OF 給与明細リスト := 0.		X		X	
A3	部合計 OF 給与明細リスト := 部合計 OF 給与明細リスト + 給与 OF 給与明細リスト.	X	X	X	X	
A4	XWRITE 合計行 OF 給与明細リスト.			X	X	

Mod0　CondExp = CB　Module->FileModifier E&O

C1	CB (残業ファイルO;<部No,社員No>)	H	B	T	E
A1	残業手当 OF 作業領域 := 0.	X			
A2	残業手当 OF 作業領域 := 残業手当 OF 作業領域 + 単価 OF 給与マスタE * 時間 OF 残業ファイルO * 掛率 OF 残業ファイルO.		X		
A3	do_again.	X	X		

図4　給与計算問題に対してEOSが生成したSPACE用のモジュール仕様群

Fig. 4 SPACE module specifications generated by EOS for the sample salary calculation problem.

従って，等関係式仕様からSPACE仕様への変換に際して行なわなければならない設計作業は以下の3つである．

①与えられた等関係式集合から計算対象が同じ等関係式をまとめてモジュールにする

②モジュールの実行状態を表わす条件式を決定する

③等関係式が表す計算の実行条件を決定する

EOS/Mはこの設計作業を自動的に行い，結果をSPACEのモジュール仕様群として生成する．

5．モジュール設計の自動化

EOS/Mによるモジュール設計とは，プロセスに対する一段修飾の等関係式仕様をSPACEのロジックテーブルで記述されたモジュール仕様に分割することである．この分割の基本は，まず与えられたプロセス仕様を構成する等関係式(が表わす計算式)集合を同一対象に対する計算式集合に分割し，これらをモジュールとする，次にモジュールの実行状態を表わす条件式を決定し，最後に等関係式が表す計算の実行条件を決定する．EOS/Mはこの設計作業を自動的に行い，結果をSPACEのモジュール仕様群として生成する．以下ではこの一連の変換手続きを示すと共に，図3に示した等関係式仕様を事例に用いて，その有効性を実証する．

5．1　SPACE構文への変換

EOS/Mが先ず最初に行うのは，等関係式仕様を構文解析し，文毎にSPACE構文へ変換することである．この段階では，識別子による修飾やファイル修飾子による修飾を取り除き，SPACEのロジックテーブル記述用の文法（条件文については5節で説明した，処理文についてはCOBOLの文法にほぼ準じる）に合うように書式を変換したり，IF文やSWITCH文を条件部分と処理部分に分割したり，式(10)に示すような漸化性を持つSUM文やFIRST文を初期処理部分と本体処理部分に分割する．

図4に示した4つのロジックテーブルは，図3の等関係式仕様からEOS/Mが実際に生成したものであり，以下ではこの事例を用いて，各変換ステップを具体的に議論する．

5．2　計算式の特性収集

EOS/Mが受け付ける等関係式は式(7)のような一段修飾の式である．この各式から，自動設計を行う指針として以下に定義する，集団識別子，単位識別子，sg特性，ファイル修飾子の諸特性を求める．等関係式(5)で，左辺の項目の識別子I0を，この等関係式gの集団識別子という．これは，式gを右辺より左辺を求める計算とみた場合の計算の範囲あるいは計算対象集団を与えている．また，EOS/Mが受け付ける等関係式は構造条件と順序条件を満足するので，与えられた仕様中に含まれる任意の2つの識別子I,Jに対する≦順序を，どれかのファイルFにおいて$I_F \leqq J_F$関係にあるということで，ファイル間の部分順序に拡張できる．従って，等関係式(7)において，右辺の引数の識別子を≦で位相ソートできる．この時，最小の識別子J(≦ ∀Ii)を単位識別子という．なお事例における識別子の順序は，残業ファイルO内での≦順序が基準になって，ファイル修飾子を無視して，<社員No,区分,@>≦<社員No,区分>≦区分≦社員No≦部Noである．

各等関係式を計算式としてみた場合に，それらを計算するのに必要なデータを用意するにはどんなファイルが必要かを考える．右辺の引数が複数ありそれらのファイル修飾子が異なる時は，式(2)からこれらの引数の値リストは同じ実体あるいは関連インスタンスのものでなければならない．この計算を行うにはそれらの実体や関連を表すファイルから照合処理で同期を取りながらレコードを入力する必要がある．このために，各等関係式に対して，その左辺および右辺の引数の識別子のファイル修飾子を全て"&"で結合し，等関係式のファイル修飾子とする．これは当然これらの識別子で指定される実体の値が全て揃わないとこの式が計算できないからである．なお，このために各式内の項目のファイル修飾子について以下の条件が満足される必要がある．満たされなければ，『等関係式...において，右左辺の項目のファイル修飾子に矛盾があります．』という理由で設計不能処理を行う．

1)項目$X_F.I_{G[K]}$において，Fは単一のファイル名である．

2)項目$X_F I_{G[K]}$において，FはGを構成する照合ファイルG=G1&G2&!G3&...内の!付きでないどれかのGiに等しい．

3)識別子の≦順序でI≦Kである．

さらに，集団識別子と単位識別子が等しい(I0=J)時，等関係式gをs(ingle)タイプといい，そうでない時g(roup)タイプという．sタイプの等関係式は，引数の値が求まった段階で，レコード読み込みを行なわずに，左辺の結果を求めることができる．sgタイプは，各等関係式の計算の実行条件を決定するために用いられる．実際，後出の表6で式が本体処理か初期処理か終了処理かに依存して，その実行条件を決定する．SUM文など式(10)に示した漸化性のある式では先に述べたように分割する．一方，基本算術式と出力式については，g特性を持つなら，明らかに本体処理とする．sタイプなら，左辺が定数なら初期処理，定数でない基本算術式や出力文なら終了処理とする．

事例に対するこれらの結果は表2のようになる．なお，右辺が定数であったり，等関係式が出力文であったりして，集団識別子か単位識別子の一方しかない時はそれらを同じものとして補うことにする．

表2　事例における各計算式の諸特性

Table 2 Characteristics of the equations for the sample problem.

式番号	単位識別子	集団識別子	sg特性	ファイル修飾子
①	<社員No,区分,@>_E&O	社員No_E&O	g	E&O[社員No]
②	社員No_E&IO	社員No_E&IO	s	E&IO[社員No]
③	社員No_E	社員No_E	s	E
④	社員No_E	社員No_E	s	E
⑤	社員No_E	社員No_E	s	E
⑥	部No_E	部No_E	s	E
⑦	<部No,社員No>_E	部No_E	g	E
⑧	社員No_E	社員No_E	s	E
⑨	部No_E	部No_E	s	E
⑩	社員No_E	社員No_E	s	E
⑪	部No_E	部No_E	s	E
⑫	@_IE&O	@_IE&O	s	IE&O[社員No]
⑬	@_IE&O	@_IE&O	s	IE&O[部No]
⑭	@_IE&O	@_IE&O	s	IE&O[部No]
⑮	@_IE&O	@_IE&O	s	IE&O[社員No]
⑯	@_IE&O	@_IE&O	s	IE&O[部No]
⑰	@_IE&O	@_IE&O	s	IE&O[社員No]
⑱	@_IE&O	@_IE&O	s	IE&O[部No]
⑲	@_IE&O	@_IE&O	s	IE&O[社員No]
⑳	@_IE&O	@_IE&O	s	IE&O[部No]

5．3　導出順序の決定と計算式の追加

データの導出関係からくる計算式間の実行順序(導出順序と呼ぶ)→を決定する．一般に2つの等関係式の引数の間に，A=f(B1,…,B,…,Bm)かつB=g(C1,…,C,…,Cn)なる関係があれば，Bを定義するgの計算が終了するまでfの計算を開始できないので，g→fとする．また，gタイプの等関係式を分割した初期処理と本体処理については，初期処理→本体処理とする．

等関係式仕様内の等関係式による項目間の導出関係が循環定義を含まないためには，各等関係式を頂点に→順序を有向辺にしたグラフが位相ソートできなければならない[無循環条件]．そうならないときは『無循環条件を満たさない』という理由で設計不能処理とする．

事例の等関係式仕様は無循環条件を満たしており，その→順序は，等関係式を番号で表すと，{①の初期処理→①の本体処理,②}→③→⑤→⑦→⑪→⑨,⑥→⑪,{④,⑤,⑥}→⑩→⑧,{⑫,⑭}→⑰→⑲,{⑬,⑮,⑯}→⑱→⑳となる．

なお，この結果推移的につく(a→b→cならa→→c)等関係式の順序を→→で表す（例：①→→⑪）．

5．4　計算式のグループ化によるモジュール作成

集団識別子とファイル修飾子が同じ等関係式は，同一のレコードグループに対する計算である．従って，これらを処理要素とするモジュールを作成する．

この集団識別子やファイル修飾子をモジュールのモジュール識別子(誤解のない時は，単に識別子)やファイル修飾子と言う．また処理要素となる等関係式が1つでもgタイプであればモジュールのsgタイプをg，そうでなければsと定義する．gタイプのモジュールは繰り返し処理を伴うグループ処理モジュールであり，sタイプのモジュールは繰り返しを行わない計算モジュールと考えられる．事例に対しては，表3のようなモジュールが作成される．なお，モジュール名は説明のために添えたもので，EOSが与える名前は単なる番号のみである．

表3　事例に対して生成されたモジュールの識別子とsgタイプ
Table 3 Identifier and sg type of the modules generated for the sample problem.

ﾓｼﾞｭｰﾙ名:番号	式番号	ﾓｼﾞｭｰﾙ識別子	sg特性	ﾌｧｲﾙ修飾子
残業計算:0	①	社員No	g	E&O[社員No]
残業なし計算:1	②	社員No	s	E&!O[社員No]
明細処理:2	③,④,⑤,⑧,⑩	社員No	s	E
合計処理:3	⑥,⑦,⑨,⑪	部No	g	E
残業社員ｴﾗｰ:4	⑫,⑭,⑰,⑲	@	s	!E&O[社員No]
残業部ｴﾗｰ:5	⑬,⑮,⑯,⑱,⑳	@	s	!E&O[部No]

5．5　モジュールの階層化

モジュール間の呼出し順序を決定するために，モジュールMを構成する等関係式の集合を{e}として，任意の2つのモジュールM1,M2に対して導出順序→を以下のように導入する．

$(\exists e1_i \in M1, \exists e2_j \in M2, e1_i \rightarrow\!\!\rightarrow e2_j)$ and $(\quad \forall e1_k \in M1, \forall e2_l \in M2, \overline{e1_k \leftarrow\!\!\leftarrow e2_l}) \Leftrightarrow (M1 \rightarrow M2) \cdots (10)$

事例に対しては，{残業計算,残業なし計算}→{明細処理，合計処理}となる．また，導出順序によって推移的につくモジュール間の順序を→→で表す時，PM→→MならPMをMの先行モジュールと言う．

さらに，任意の2つのモジュールM1,M2に対して呼出し可能順序➡を以下のように導入する．すなわち，M1の任意の直和分割$M1^f+M1^l$に対して，

$(M1^f \rightarrow M2 \rightarrow M1^l) \Rightarrow (M1 \Rightarrow\!\!\blacktriangleright M2) \cdots (11)$

これは，M1がその前半$M1^f$と後半$M1^l$に直和分割でき，導出順序で$M1^f$，M2，$M1^l$に並べることができるならば，M1内の命令を$M1^f$，EXEC M2，$M1^l$の順に並べれば導出順序が満たされることを意味している．なお，$M1^l=\phi$や$M1^f=\phi$のような特殊な場合には，式(12)に示すように，EXEC M2をM1の最後で行ったり最初で行うことができる．

$(M1 \rightarrow M2) \Rightarrow (M1 \Rightarrow\!\!\blacktriangleright M2)$ and $(M2 \Rightarrow\!\!\blacktriangleright M1) \cdots (12)$

事例に対しては，明細処理の直和分割{③,④,⑤}+{⑧,⑩}に対して，{③④,⑤}→合計処理={⑥,⑦,⑨,⑪}→{⑧,⑩}なので，明細処理➡合計処理が言える．

以上の諸概念を踏まえて，モジュール間にサブルーチン呼出しの関係$\Rightarrow\!\!\blacktriangleright_e$とコルーチン呼出しの関係$\Rightarrow\!\!\blacktriangleright_d$を以下の手順で導入し，モジュール集合を階層的に構造化する．

[i]モジュールグラフへの分割

ファイル修飾子が同じかあるいは少なくとも1つのファイル名を共有する(この際FiとIFiを同一視する)モジュールを集めてモジュールグラフを作成する．これによってモジュールをいくつかのモジュールグラフMG(F)に分割する．なお，この時のF=F1&F2…はモジュールグラフ内に現れるモジュールのファイル修飾子内の全てのファイル名Fiを&で結合したものである．

[ii]モジュールグラフ内に呼出し順序による辺の導入

a)MG(F)のFが単一のファイル名からなる場合(計算／グループ処理モジュールのみの階層化)

a-1)モジュールM.I_FとM'.I'_Fの間に，I>Iなら有向辺M'.$I'_F \Rightarrow\!\!\blacktriangleright_e$M.$I_F$を導入する．なお，5．4節で同じ集団識別子を持つ式をモジュールにしているので，識別子間に等号関係は成立しない．

a-2)モジュールグラフMG(F)を$\Rightarrow\!\!\blacktriangleright_e$順序で線形ソートする．一番大きい識別子を持つものをトップモジュールTとする．線形ソートできない場合は「同一の単一ファイル修飾を持つモジュール間に線形順序が成立しない」という理由で設計不能処理とする．

b)MG(F)のFが複数のファイル名からなる場合(照合処理モジュールの追加を伴う階層化)

b-1)複数のファイル名からなるKを照合キーとしたファイル修飾子G[K]⊂Fを持つモジュールM.$I_{G[K]} \in$ MG(F)があれば，Kを照合キーにしてFを構成する全ファイルを対象とした照合モジュールMC(F,K)を作成する．さらに作成された照合モジュールMC1,MC2の識別子が>関係にありその間に同様の他の照合モジュールが存在しないなら，有向辺MC1$\Rightarrow\!\!\blacktriangleright_e$MC2を導入する．また，最大の識別子を持つ照合モジュールをトップモジュールTとする．

b-2)非照合モジュールM.$I_{G[K]} \in$ MG(F)を→順に，表4に従ってMG(F)内の照合モジュールやこれらに同様にして既に連結された非照合モジュールに連結する．

表4　モジュールM.IGの連結先の決定
Table 4 The rules to decide where to connect module M.$I_{G[K]}$

モジュールM.$I_{G[K]}$のsgﾀｲﾌﾟ=s	Y	Y	N	N	N	N	N	N
先行モジュールを持たないか，先行モジュールを呼出している*1	–	–	Y	Y	Y	Y	N	N
I>Iなる識別子Iを持つｸﾞﾙｰﾌﾟ処理ﾓｼﾞｭｰﾙM'.$I'_{G[K]} \in$MG(F)がある	Y	N	Y	N	–	–	–	–
先行モジュールを呼出す照合モジュールMC(F,J)の最小識別子J<I	–	–	–	–	–	Y	Y	N
G[K]を構成するファイル名が全て付までない	–	–	Y	Y	N	Y	N	–
有向辺M'.$I'_{G[K]} \Rightarrow\!\!\blacktriangleright_e$M.$I_{G[K]}$を導入する．	a)		b)					
有向辺（あればMC(F,I)，なければMC(F,K)）$\Rightarrow\!\!\blacktriangleright_e$M.$I_{G[K]}$を導入する．			c)		d)	e)		f)
有向辺MC(F,K)$\Rightarrow\!\!\blacktriangleright_d$M.$I_{G[K]}$を導入する．							g)	
「付きﾌｧｲﾙ修飾ｷｰﾜｰﾄﾞの呼出し場所を決定できない」ｴﾗｰ．								h)

*1:直接的にあるいは間接的に呼出している

ただし，この$\Rightarrow\!\!\blacktriangleright_e$順序を導入する際には注意が必要である．表4で(a)と(b)においては，呼出し可能順序M'➡Mが成立するならよいが，そうでない時は『M'からMを呼出すことは，導出順序と矛盾するのでできません』という理由で設計不能処理を行う．しかし，(c)から(g)においては呼出しモジュールMCは照合型であり，それに所属する命令は下位モジュールの呼出し命令であり，その間の実行順序は下記の5．9節で調節可能である．

[iii]トップモジュールの確定

モジュールグラフが1つしかなければ，このモジュールグラフ内のトップモジュールを最終的なトップモジュールとする．そうでなければ，モジュールグラフ間にモジュール間の→順序を拡張し，これによってモジュールグラフの→順でそれぞれのトップモジュールを順に呼出すモジュールTを作成し，これを最初に実行するトップモジュールとする

事例では，表3に示した6つのモジュールはOかEをファイル修飾子内に共有しているので，MG(E&O)を作成する．従って手順[ii]のb)より，部照合モジュールMC(E&O,部No):Mod7と社員照合モジュールMC(E&O,社員No):Mod6を作成し，辺MC(E&O,部No)$\Rightarrow\!\!\blacktriangleright_e$MC(E&O,社員No)を付ける．次にMG(E&O)内のモジュールを→順に位相ソートし，{残業計算,残業なし計算}→{明細処理，合計処理},残業部エラー,残業社員エラーの順に，それぞれ表4のd),(c),(c),(g),(c),(c)に従って，辺$\Rightarrow\!\!\blacktriangleright_e$で図5の様に他のモジュールに連結される．なお，これらの連結は全て呼出し可能順序➡を満たしており設計可能である．なお，トップモジュールは部照合モジュールとなる．

この呼出し順序の導入は照合モジュールを中心に，それより大きいキーのグループ処理モジュールはSPACEのLC条件式を用いて作成しそれをコルーチン$\Rightarrow\!\!\blacktriangleright_d$として呼出すことを意味し，それより小さいキーに対するグループ処理モジュールはSPACEのCB条件式を用いてサブルーチン$\Rightarrow\!\!\blacktriangleright_e$として呼出すことを意味している．このようにして，作成されたモジュール群がトップモジュールを頂点にして階層化され，モジュール設計が完了する．

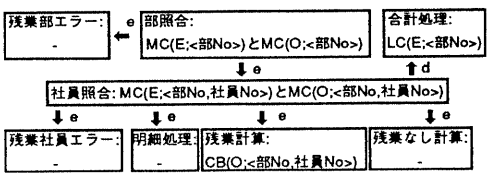

図5　事例に対するモジュール階層と条件式
Fig. 5 Module hierarchy and condition expressions generated for the sample problem.

5．6　モジュールの制御論理の決定

先に述べたように，SPACE仕様においてはモジュールの制御論理は条件欄に記入する条件式の組合わせとして表される．各モジュールに対する条件式は表5に示す規則に従って決定される．一般に，一段修飾のEOS仕様はEOS/Pによる設計の結果構造条件を満足している．従って，ファイル修飾子が複数のファイル名$F_1,F_2,..F_i,..$から構成されている場合，これらのファイルは共通の識別子Iに対して≦順でそれより大きい識別子I0,I1,..を全て共通に持っている．従ってこれらの識別子を全て大きい順に繋げた項目<I0,I1,...,I>を識別子Iに対応するキーと呼び，key(I)と書く．これを照合キーとして用いて，照合モジュールではn個のファイル$F_1,F_2,..F_i,..$に対するMC式を表5に示すように生成する．

一方，グループ処理モジュールでは，ファイル修飾子が複数の!付きでないファイルF1,F2,...からなっている時は考慮が必要である．なぜなら，複数のファイルからのレコード入力を伴うグループ処理をSPACEの1つのモジュールで表現することは不可能であるからである．従って，この様なグループ処理モジュールでSPACEのモジュールに展開できるのは次の2つの場合のみである．

1)！付きでないファイルFが1つの場合．

この場合は，事実上ファイルFのみに関するグループ処理モジュールとして，他のファイルGなどは無視して（すなわちGからのCBやLCは作成しないようにして）表5に従って制御論理を決定する．

2)!付きでないファイルF1,F2,...が複数あったとしても，この時のモジュール識別子をIとして，ファイルF1,F2,...のうち唯1つのファイルFのみにおいて@<Iの関係にあり，他のファイルでは@=Iの関係にある場合．

この場合は，実際に読み込むファイルはFだけに限定され，他のファイルのレコードは既に上位モジュールで読み込まれているレコードを用いて計算可能である．従って，このファイルFのみに着目して表5に従って制御論理を決定する．これらの2つの場合以外の時は，このモジュールに関しては，「2つ以上のファイルに対するグループ処理モジュール」という理由で設計不能とする．なおこの時，!付きのファイルをファイル修飾子から削除してはいけない．なぜなら，表6におけるdo MやexecMによるモジュールMの呼出し命令の実行条件の決定において，ファイル修飾子内の!付きのファイルも考慮の対象となるからである．

この方法で決定される条件式は，モジュールが行うレコードの入出力制御を決定するものだけである．もしモジュールに割り当てられた等関係式内にswitch文やif文があれば，データ項目の値による分類を行う条件式を条件部に挿入する．結局事例に対しては，図5で：に続けて示したように各モジュールの条件式が決定される(特に条件式がない場合は-で示した)．

表5　モジュールの条件式と追加命令の決定方法

Table 5 The rules to decide condition expressions of a module and action statements added to the module.

型は	呼出し辺	条件式	追加命令
照合	-	n個のMC(Fi;key(I))	do_again
ｸﾞﾙｰﾌﾟ	$\Rightarrow_e$	CB(F;key(I))	do_again
ｸﾞﾙｰﾌﾟ	$\Rightarrow_d$	LC(F;key(I))	なし
計算	-	-	なし

5．7　処理命令の追加

モジュール内で計算を繰返し行うか行わないかは，モジュールがトップモジュールであるかどうか，モジュールに割り当てられた計算がsタイプかgタイプか，などによって決まる．結局これは，表5のように整理され，必要に応じて"do_again"文を処理部に追加することになる．

さらに，モジュールグラフにおけるモジュールAからモジュールBへの辺ごとに，それが$A\Rightarrow_e B$ならサブルーチン呼び出し"do B"文を，$A\Rightarrow_d B$ならサブルーチン呼び出し"exec B"文を，Bを呼び出す命令としてモジュールAの処理部に追加する．

ところで，グループ処理モジュールが表4の(a)や(b)のように階層的に呼出されている場合，下位モジュールの繰り返し中に上位の識別子による制御切れが起こった時，上位モジュールに戻ることは，後出の表6に示すようにdo_againをT条件値の時に行わないことによって実現する．しかし，一方ファイル$F_1,F_2,..F_i,..$に対するキー<I0,I1,...,In,I>による照合モジュールでは，SPACEのMC条件式が純粋に照合条件のみを認識するので，Iより大きい識別子I0,I1,...による制御切れを条件値として認識できない．従って各ファイルF_i(そのファイル番号をniとする)と各上位キー<I0,I1,....,Ij>(ただしj<n)(そのキー番号をkjとする)とこれを照合キーにする上位の照合モジュール(そのモジュール番号をmkとする)に対して，以下のようなif文を作成し前処理欄に追加する．

IF DTki--MC--kj NOT = Fni--kj--C THEN EXIT.

事例では，図4の社員照合モジュール:Mod6に前処理P1,P2が追加される．

5．8　モジュールの統合

sタイプのモジュールMで呼び出し親のモジュールNと同じモジュール識別子を持つものは，その引数の値が確定した段階で，レコードの読み込みを行なわずに計算を完了できる．従って，このモジュールを呼び出す親モジュール内に，その呼び出し処理の代りに，このモジュールの処理要素を直接展開することにする．なお，この際統合されるモジュールMから出ている辺を統合先の親モジュールNとの辺に取り替える必要がある．

このモジュールの統合においては，条件式は単に条件欄において併合するだけでよい．事例では，図5において明細処理モジュールと残業なし計算モジュールと残業社員エラーモジュールを社員照合モジュール内に展開する．また，残業部エラーモジュールを部照合モジュールに展開する．

5．9　処理命令の整列と計算条件の決定

最終段階として，モジュールに割り当てられた計算式の実行条件を決める．まず，ロジックテーブルの条件部にこれまでに求まった条件式を記入する．次に，これらの条件式の相異なる値の組合せであるすべての規則列を生成する．一方，処理部には，モジュールに割り当てられた全ての計算式(モジュールの呼出し命令やdo-againなどの追加処理文も含む)を，これらに対する導出順序→の順に記入する．ただし，モジュールの呼び出し命令については，呼び出されるモジュールの処理要素との導出順序を考慮する．

具体的には，$M1\Rightarrow_e M2\Rightarrow_e M3\Rightarrow_e..$のようにモジュールが階層的に呼び出されている時，モジュールMとそれより下位の全てのモジュールに属する式の和集合をbody&sub(M)と書く(例：body&sub(M2)=M2∪M3∪...)．Mの呼出し命令を除く全ての処理命令を{m}，Mが直接呼出しているモジュールをN1,...Ni,...として，{m1,...,mj}→body&sub(Ni)→{mj+1,...}なる場所を見つけて，{m}を{m1,...,mj,EXEC Ni(またはDO Ni),mj+1,...}に置き換える．また，do-againは最後に記入する．さらに，呼出し順序が決定したらFOLLOW文は取り除く．

具体的には，$M1\Rightarrow_e M2\Rightarrow_e M3\Rightarrow_e..$のようにモジュールが階層的に呼び出されている時，モジュールMとそれより下位の全てのモジュールに属する式の和集合をbody&sub(M)と書く(例：body&sub(M2)=M2∪M3∪...)．Mの呼出し命令を除く全ての処理命令を{m}，Mが直接呼出しているモジュールをN1,...Ni,...として，{m1,...,mj}→body&sub(Ni)→{mj+1,...}なる場所を見つけて，{m}を{m1,...,mj,EXEC Ni(またはDO Ni),mj+1,...}に置き換える．また，do-againは最後に記入する．さらに，呼出し順序が決定したらFOLLOW文は取り除く．

事例では，例えばMとして部照合モジュールを考える．ここには，EXEC社員照合モジュール,⑬,⑮,⑯,⑱,⑳,do-againがある．body&sub(社員照合モジュール)={全ての他の式}．{⑬,⑮,⑯,⑱,⑳}は他の式との間には→順序がないので，例えば，EXEC社員照合モジュールの後にして，FOLLOW文の⑱は除いて結局図4のMod7の様に整列される．また，6.5節で議論したように明細処理の直和分割{③,④,⑤}+{⑩,⑧}に対して明細処理⇒合計処理とできるので，社員照合モジュール=

{{body&sub(残業計算)={①の初期処理→①の本体処理},②}→{③,④,⑤}→body&sub(合計処理)={⑥,⑦,⑨,⑪}→{⑩,⑧},body&sub(残業社員エラー)={⑫,⑭,⑰,⑲}}より，図4のMod6の様に整列される．

最後にこれら各計算式が，モジュールのどんな状態の時に実行されるかを決定する．この決定ルールを，表6と表7に示した．表6は条件部に記入された条件式が入出力制御に関するMC,CB,LCの場合を表し，表7は分類条件の場合を表している．具体的にはこれらの表は，『処理部に記入された計算式がこの表の右欄に示すどの特徴を持つかによって，条件部に記入された各条件式がこの表の左欄に示すタイプを持つ行の中央欄に記入された値をとる規則列に，実行指示記号Xを入れる』ことを指示している．

この結果事例に対しては，先に図4に示した4つのSPACEのモジュール仕様が生成される．この仕様は5節で説明した様に確かに図1に示した給与計算を行う．この設計結果は人手によって設計した結果と，読みやすさのために計算式のモジュール内での位置が異なる点などを除いて，論理的には全く同じものであった．なお，図1における処理文とその元となった等関係式との対応は，例えばMod6のA2←②,Mod0のA1←①の初期処理,Mod0のA2←①の終了処理などであるが，これらは式の内容から容易に分るであろう．

特にレコードの入力制御に関しては，部照合モジュール(Mod7)がトップモジュールになり<部No>キーによる照合処理を繰返し行い，この中でさらに下位キーである<部No,社員No>キーによる社員照合モジュールが呼出されている．社員照合モジュールから呼出される部合計モジュール(Mod3)は，給与マスタの<部No>キーに関する集計処理を行うが，親モジュールが<部No,社員No>キーに関する繰り返しを行い，しかも<部No>キーの方が<部No,社員No>キーより大きい集団を識別しているので，レコードごとに状態を把握できるLC式を条件式として用いるように設計されている．さらに処理としてdo_againを使わないで，モジュール内での繰り返しを行わないサブルーチンになるように設計されている．同様に呼出される残業計算モジュール(Mod0)は<部No,社員No>キーによる繰返しなので，CB式を条件式に用いた設計となっている．また，内部でdo_againによって繰返しを行いサブルーチンとして呼出されるように設計されている．

サブルーチンを巧く使った設計が生成されるので，中間ファイルを用いず1パスで処理され実行時間は短くなる．このように，EOSを使えば高度な設計テクニックを用いたSPACE仕様が自動的に生成されることを示している．

表6　モジュール内の計算式の実行条件の決定方法1(入力制御に関する条件式について)

Table 6 Decision rules for the execution conditions of computation under file input control.

条件式	実行すべき計算式	条件値
MC(F;K)	ﾌｧｲﾙ修飾子*1がFを含む	Y
MC(F;K)	ﾌｧｲﾙ修飾子がFを含む	N
MC(F;K)	do_again	Y*3,N
CB(F;K)	初期処理	H
CB(F;K)	本体処理	B
CB(F;K)	終了処理	T
CB(F;K)	do_again	H,B,T*2
LC(F;K)	初期処理	F,U
LC(F;K)	本体処理	F,I,U,L
LC(F;K)	終了処理	U,L

*1:式gの集団識別子，あるいは呼び出し命令の呼び出し先モジュールの識別子の，ファイル修飾子．

*2:これがトップモジュールの時のみ．

*3:do-againより前に照合モジュールMの呼出しEXEC M.を実行していれば，Xはいれない．

表7　モジュール内の計算式の実行条件の決定方法2(分類に関する条件式について)

Table 7 Decision rules for the execution conditions of computation under value dependent branching control.

条件式	条件値	実行すべき計算式
S={…;Si;…}	i	S=Siに対応する計算式
論理式C	Y	C=Yに対応する計算式
論理式C	N	C=Nに対応する計算式

6．　おわりに

本研究によって，等式集合としてプロセスに対する非手続き的に与えられた仕様から，計算の対象の包含関係やデータの導出順序をもとに，プロセスをモジュールに分割し，各モジュールの正確な仕様を決定するモジュール設計の理論を構築した．さらにこの理論を実証するシステムEOS/Mを作成した．事例に適用したところ，熟練者と同程度の良好なモジュール仕様を生成できた．また，このモジュール仕様から自動プログラミングSPACEによってCOBOLプログラムが生成され，事例に要求された計算が実行された．この結果，ファイル処理に関して，プロセスに対するモジュール設計を自動的に行うという目標は達成された．

今後の課題としては，SPACEの設計インタフェースに等関係式仕様の編集ウインドウを追加する，などの統合化が残っている．これによって，EOSとSPACEを完全に結合することができ，一貫した自動開発を実現できる．また,理論的にこの自動設計過程が正しいことを形式的に証明することなども重要と考えている．

謝辞

本システムの開発に協力してくれた原田研究室出身の中村義幸(現ソニー㈱)神山幸一(現㈱東芝)，浅見伸美(現㈱東芝)，前島康伯(現㈱日立製作所)の諸氏に感謝する．

参考文献

1) Barstow,D.:Domain-Specific Automatic Programming,IEEE TSE-11,No.11,pp.1321-1336(1985).

2) Balzer,R.:Operational Specification as the Basis for Rapid Prototyping,ACM SIGSOFT,SOFTWARE ENGINEERING NOTES,Vol.7,No.5,pp.3-16(1982).

3) Chen,P.P.:The Entity-Relationship Model-Toward a Unified View of Data, ACM Trans. on Database Systems,Vol.1,No.1,pp.9-36(1976).

4) Darlington,J.:An Experimental Program Transformation and Synthesis System,Artificial Intelligence,Vol.16,pp.1-46(1981).

5) 原田実,篠原靖志:部品合成によるﾌﾟﾛｸﾞﾗﾑ自動生成ｼｽﾃﾑARIES/I,情処学論,Vol.27,No.4,pp.417-424(1986).

6) 原田実:事務処理分野における自動ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ,情報処理,Vol.28,No.10,pp.1378-1397(1987).

7) 原田実:COBOLﾌﾟﾛｸﾞﾗﾑ自動生成ｼｽﾃﾑSPACEによる仕様の視覚化と抽象化,信学論,VolJ71-D,No.7,pp.2555-2562(1988).

8) 原田実,中村義幸:ﾌﾟﾛｸﾞﾗﾑの構造と論理の自動設計ｼｽﾃﾑEOS/M,情報処理学会論文誌,VOL.34,No.9,1993,2013-2024頁(1993).

9) 原田実,西村淳一,中村義幸:非手続き仕様からのﾌﾟﾛｾｽ設計の自動化,電子情報通信学会論文誌D-I,VolJ77-D-I,No.2,pp.196-206(1994).

10) 河野史男他:ﾃﾞｰﾀに着目した仕様を入力とするCOBOLﾌﾟﾛｸﾞﾗﾑ生成ｼｽﾃﾑDSLの開発,日立評論,Vol.65,No.7,pp.53-58(1983).

11) Manna,Z. and Waldinger,R.:A Deductive Approach to Program Synthesis,ACM TOPLAS,Vol.2,pp.90-121(1980).

12) Prywes N.S. and Pnueli A.:Compilation of Non-procedural Specifications into Computer Programs,IEEE Trans. Software Eng.,Vol.SE-9,No.3,pp267-279(1983).

13) Ruth G.:ProtosystemI: An automatic programming system prototype, Proc. of the NCC,Anaheim,Calif.,AFIPS 47,pp.675-681(1978).

14) 杉山健司他:対話型自然言語ﾌﾟﾛｸﾞﾗﾐﾝｸﾞｼｽﾃﾑの試作,信学論,VolJ67-D,No.3,pp.297-304(1984).