TM-0480

制御分野における自動プログラミング

本位田真一，内平直志，中村英夫

March, 1988

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191-5
Telex ICOT J32964

「情報処理」第28巻 第10号別刷 昭和62年10月発行

制御分野における自動プログラミング

本位田 真一　内平直志　中村英夫

社団法人 情報処理学会

Vol. 28 No. 10　　情　報　処　理　　Oct. 1987

解　説

3. 応用分野の最前線

3.4 制御分野における自動プログラミング†

本位田 真一††　内平 直志††
中村 英夫††

1. はじめに

電力系統や発電所などのプロセス制御から，ロボット制御，数値制御，シーケンス制御などに至るまで，計算機による制御対象は多岐にわたっているが，共通な課題として，高品質，高信頼性なソフトウェアをいかに効率良く生産するかということが認識されており，従来からおのおのの分野において活発な研究が行われてきている．それらの研究の位置付けは大きく次の2つに分類される．

① 特定の制御分野に限定して，その制御分野固有の問題として捉えている．

② 一般に計算機による制御システムにおいては，制御対象が主に機械，電気系であるため，外界に対しての迅速な応答性，高信頼性が要求されている．特に応答性の向上および外界である制御対象の自然なモデル化などのために並行プロセスの形態を成している場合が多いが，その結果，並行プロセスの検証の困難さという大きな課題も生じている．したがって特定の制御分野に依存せずに，一般の並行処理における固有問題として捉えている．

このように，ひとくちに制御分野における自動プログラミングと称してもその内容が大きく異なっている．分野を限定した自動プログラミングについてはおのおのの分野において問題向き言語や支援ツールが実用に供されている．一方，特定の分野に依存しない自動プログラミングは，具体的には並行プロセス間の同期や排他制御問題や，デッドロックなどの問題に帰着するが，商用化には至っておらず，研究レベルにとどまっている．

そこで本稿では，これらの2つの動向を踏まえて，まず並行プロセスである制御用プログラムの特性を示し，並行プロセスにおける機能部分と同期部分の相違点を明確にし，次に上記①と②のおのおのについての自動プログラミングについて現状の動向を概観し，最後に今後の方向について触れる．

2. 制御用プログラムの特性

発電所やロボットなどの被制御装置を対象とする制御用プログラムは一般的には組込み型システムにおけるソフトウェアであり，次のような特徴を有している[1]．

① いわゆる計算自体が主目的ではなく，外界のモニタリング，外界への制御信号の出力，オンライン・データベースの管理などが主要な機能である．

② 外界における状態，計算機側における状態があり，時間の経過とともにそれぞれが互いに干渉しながらそれぞれの状態を遷移していく形態をとっているためこれらを記述する必要がある．

③ 外界からの複数の信号入力に対しての迅速な応答を可能とするためにリアルタイム OS の下での並行プロセスの形態を成している．

④ 計算機側と外界とのやりとりが行われるので一般にテストが困難である．

⑤ 外界を制御するために誤動作をしない fail-safe な信頼性が要求される．

⑥ 性能に関しての要求が厳しい．ここでの性能とは，被制御機器からの入力信号に対する処理時間である応答時間に相当するものである．

上記のように制御用プログラムは効率などの点から並行プロセスの形態をとっているが，その中で並行処理の単位はタスクやプロセスなどと呼ばれている．一方，タスクやプロセスの中身は逐次的に実行する．別の見方をすれば，並行プロセスである制御用プログラムは大きく分けて

- 機能に関する部分

† Automatic Programming for Control Systems by Shinichi HONIDEN, Naoshi UCHIHIRA and Hideo NAKAMURA (Systems and Software Engineering Laboratory, Toshiba Corporation).

†† (株)東芝システム・ソフトウェア技術研究所

●同期に関する部分

に分類できる[3]．したがって，機能に関する部分はタスクやプロセスの中身に相当し，同期に関する部分はタスクやプロセス間の関係を示すことになる．

制御用プログラムにおける自動プログラミングの目的は上の2点において異なっている．すなわち，機能部分の生成においては所望のプログラム量が多いために高生産性が要求されるのに対して同期部分の生成においては，少量であるが特に高信頼性が要求される．具体的には同期部分に関しては，機能の動的関係，たとえばデータの受け渡しの正しさ，デッドロック・フリー，共有データ・アクセス，同期などの検証項目を満たす必要がある．

また種々の分野における制御用プログラムにおいて，機能に関する部分はおのおのの分野に依存している．したがって機能に関する部分の自動プログラミングは必然的に分野ごとに考えざるをえず，逆に分野を限定し十分狭い領域を設定することにより，かなりの部分の完全自動化が可能になることは明らかである．一方，対象領域を広く設定した場合には，要求仕様からプログラムまでのプロセスを完全自動化することは困難であり，現在の技術レベルでは，そのプロセスをいくつかの段階（たとえば設計仕様からプログラム）に分けた処理ステップの1つを自動化すること（部分的自動化[3]と呼ぶ）にとどまっている．同期に関する部分は並行プロセス間の問題として分野に独立に考えることができるが，並行プロセスの自動プログラミングにおける部分的自動化の1つとして位置付けられる．

このように，並行プロセスにおける機能に関する部分は分野を限定した自動プログラミング，および同期に関する部分は並行プロセス間の自動プログラミングとみなせるがおのおのにおいて，それらの形態が大きく異なっている．

3. 分野を限定した自動プログラミング

分野を限定した制御用プログラムに対する自動プログラミング手法は，基本的には問題向き言語（POL：Problem Oriented Language）の形態をとっている．POL によって生成されるプログラムの多くは主として並行プロセスの機能に関する部分に相当する（プロセス制御の分野では，同期に関する部分も合わせて生成するシステムも一部存在する）．POL の多くは，空欄記入方式あるいはメニュ方式であり，その目的は，プログラミングの知識がなくても，その対象分野の知識のみで，プログラムを生成できることである．特にいわゆる高級言語であると，その言語を習得する必要があるという問題が生じる．当該分野のユーザが必ずしもソフトウェア技術者には限定されないため，言語の習得は現実的ではない．したがって，おのおのの分野において，その特徴を生かした POL が発表されている．本章では，制御分野としてロボット制御，プロセス制御，シーケンス制御，数値制御の分野を選択し，POL の現状を紹介する．

3.1 ロボット制御

ロボット制御の分野ではプログラム生産性の向上のために，次の2つの手法が採用されている．

① Teach Pendant Programming

② High Level Language Programming

Teach Pendant Programming（人間がアームなどの動作をあらかじめ教示する手法）は現在最も使われており，アームの動きや，単純な逐次型プログラムのプログラミングには適しているが，Teach Pendant から，複雑なロボット制御プログラムを生成するのは容易な作業ではない．その結果，信頼性に欠けるプログラムを生成することがあるという欠点がある．一方，High Level Language には，多くの商用化された言語があり，Teach Pendant Programming に比較してより複雑なアプリケーションが記述できるが，その言語を習得しなければならないという問題がある．そこでこれらの2つの手法の中間点にある手法として Menu Programming が提案されている[4),5)]．Menu Programming は H. Gomma によって提案されている手法であるが，ロボット特有の操作コマンドや WHILE loop, IF-THEN などのプログラム構造に関するコマンド，Teaching のためのコマンドなどがメニュとして用意されている．ユーザはこれらのコマンドにより，特に言語を習得することなく会話形式でのプログラミングを行うことが可能となる．

3.2 プロセス制御

外界のプロセスを制御する計算機システムのことをプロセス制御システムと呼ぶ．対象となるプロセスは，電力系統や鉄鋼の圧延システムなどの工業プロセスから，宇宙などの移動物体などであり，リアルタイムシステム[6]と呼ばれるものである．これらのプロセス制御システムにおいても従来より自動プログラミングの手法は積極的に採用されてきた．具体的には，対象システムの制御手順，制御条件，制御操作に関する

発した[2]．その手法を図-1に示す排他制御の例によって概説する．この例は，critical section に相当するプロセスSとそのユーザプロセスであるP1とP2において，P1とP2が互いにSを同時に占有することなく，非同期に動く問題である．P1とP2の動きは，PTLを用いて次のように記述できる．

$$S!\mathrm{begin}_i$$
$$\wedge\Box(S!\mathrm{begin}_i \supset \bigcirc S!\mathrm{end}_i)$$
$$\wedge\Box(S!\mathrm{end}_i \supset \bigcirc S!\mathrm{begin}_i)$$

ここで $S!\mathrm{begin}_i$ は $Pi\ (i=1,2)$ から critical section への entry に関する要求を表している．同じように，$S!\mathrm{end}_i$ は critical section から出ることを要求している．また，2番目の式は，$S!\mathrm{begin}_i$ の次の要求が $S!\mathrm{end}_i$ であることが常に成立することを示している．一方，Sの仕様は次のように表現できる．

$$\Box(P1\ ?\ \mathrm{begin}_1 \supset([\neg P2\ ?\ \mathrm{begin}_2] U[P1\ ?\ \mathrm{end}_1]))$$

これは，P1がcritical sectionに入ったら，P1がそのcritical sectionから出るまでP2がcritical sectionに入ることを許さないことを示している．これらの与えられた仕様からWolperは定理証明手法の1つであるタブロ法を用いてCSPプログラムを生成している．タブロ法とは，与えられた論理式が恒真であるかを証明するために，図-2に示すdecomposition ruleを用いて，その論理式を現在と次の時点以降に分割する．たとえばある論理式 F を次のように A と B に分割し図-3に示すようなグラフを生成する．

$$F \rightarrow A \wedge \bigcirc B$$

このグラフにおいては A が現在真となる論理式であり，B が次の時点で真となる論理式である．さらに，B に対しても同じ分割を行うことを繰り返し，グラフを生成する．最後にeventualityが満たされているか

$$\Box f = f \wedge \bigcirc \Box f$$
$$\Diamond f = f \vee \bigcirc \Diamond f$$
$$f_1 U f_2 = f_1 \vee (f_1 \wedge \bigcirc(f_1 U f_2))$$

$\bigcirc f$（次の状態で f が真である）
$\Diamond f$（将来 f が真である）
$f_1 U f_2$（f_2 が真になるまで常に f_1 が真である）
$\Box f$（常に f が真である）

図-2　decomposition rule

図-3　タブロ・グラフ

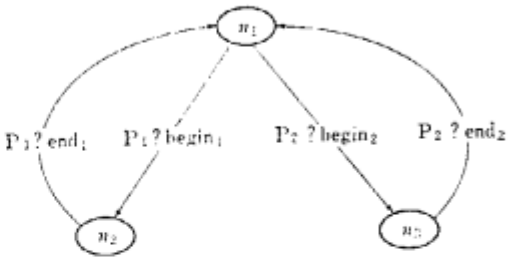

図-4　S に対するタブロ・グラフ

```
N=1;
*[N=1; P1?begin1 → N=2
 □N=1; P2?begin2 → N=3
 □N=2; P1?end1 → N=1
 □N=3; P2?end2 → N=1]
```

図-5　S に対するCSPプログラム

の吟味を行い（満たされないノードを削除する），その結果，グラフが空になれば，F が充足不可能であることがいえる．F が充足可能ならばグラフが生成されるが，そのグラフは状態遷移図であるため，プログラムの実行順序を表現しているとみなすことができる．その実行順序はたとえばCSPで表現することは可能である．排他制御の例の S に対しては図-4に示すグラフが生成され，このグラフから図-5に示すCSPのプログラムが生成される．このプログラムはまずグラフの位置（$N=1,2,3$）をチェックし，対応する入力を待ち，結果としてグラフの位置を更新することを繰り返すことを示している．

さらに，WolperはPTLと正規文法（RG：Regular Grammar）を融合したETL（Extended Temporal Logic）に対してもPTLに対してのと同様の手法でプログラムを合成することを提案している．RGで受理される記号の系列 $a_0, a_1, a_2, \cdots$ において記号 a_i を時刻 i において論理式 a_i が真であるとみなすと，RGとPTLの接点が生じる．たとえばPTLの式 $\Box a$ と生成規則が $\{S_0 \rightarrow aS_0\}$ であるRGは等価である．一般にPTLとRGの記述能力はおのおの，特色があり，いずれかが，いずれかを含むというものではない．そこでPTLとRGを統合し，記述能力を増した論理がETLである．すなわちETL＝PTL＋RGである．ETLの記述力は，PTLに命題変数とそれに対する限定記号を付加したQPTL（Quantified Propositional Temporal Logic）と同等であるにもかかわらず，その決定手続きの領域計算量がPSPACE完全でありQPTLよりも計算コストが小さいため有用性は高いと思われる．

また，ClarkeとEmersonは並行処理システムの形

式的仕様として時制命題論理を用いており，タブロ法により状態遷移図を生成する手法を提案している[13]．同期部分の生成手法としては Wolper の手法に近い．大きな相違点は，Wolper の手法では線形時間を対象としているのに対して Clarke と Emerson の手法では分枝時間体系であるところである．分枝時間体系を採用した目的は，ある性質を満たす時系列の存在を陽に表現することであるとしている．Clarke と Emerson の分枝時間体系における決定手続きの計算量は EXPTIME 完全であり，線形時間体系に対して計算コストは高いが，記述力の点で前者が後者を含んでいることも証明されている．

論理式からの定理証明手法は与えられた論理式の充足可能性の決定手続きを利用している．したがって，生成されたプログラムは正しいことが保証される．しかし，論理体系が複雑になると，それにともないその決定手続きの計算量が膨大になる[14]．たとえば線形の時制命題論理では，PSPACE 完全であるが，QPTL では non-elementary，local ITL (Interval Temporal Logic) でも non-elementary になることが知られている[15]．このように，ある程度定理証明手法を実用に供するように利用するためには入力となる仕様記述の範囲を制限しなければならない．当然のことながら，命題論理のみでは，記述可能な仕様の範囲にも制限が生じてくる．また，所望のプログラムをすべて定理証明手法で生成するのは実用的ではない．そこで，現在の技術レベルでは，定理証明手法にすべて依存するのではなく，他の手法との組合せが現実的である．その1つの手法としては部品の再利用との組合せが考えられる．すなわち，多くの部分には部品（並行プロセスの機能部分に相当する）を積極的に用いて，部品間のインタフェース（並行プロセスの同期部分に相当する）に定理証明を用いる手法である．特に対象が並行処理システムの場合には，部品の実行順序によってはデッドロックを起こしたりするため，その順序を規定する必要がある．そのような順序の規定に定理証明手法を用いようとするものである．

その手法の事例として，著者らが開発中のシステムを紹介する[16]．このシステムは並列型オブジェクト指向言語 MENDEL/87[17] を対象としている．MENDEL/87 では通信パイプと呼ぶチャネルを経由してオブジェクト間のメッセージを伝達させる．すなわち，MENDEL/87 においては，オブジェクトが部品，通信パイプが部品間のインタフェースにおのおの対応する．ここで各通信パイプの中央にはゲートが付随しており，たとえメッセージが通信パイプに送り出されても，ゲートが開いていなければ相手先にはメッセージが届かないことになる．これらのゲートを管理するのはゲート・コントローラと呼ばれるものである．すなわち，ゲートの開閉に関する仕様を PTL で与え，定理証明手法によってゲート・コントローラに対する入力データを生成する．この手法は Wolper の手法に基づいているため，ゲート・コントローラは生成される状態遷移図に対するインタプリタの形式となる．

簡単な例で説明する．あらかじめ部品として次の3つのオブジェクト（部品）が用意されているとする．

○単語切り出しオブジェクト

入力される文字ストリームから単語を切り出して出力する．

○キーワードチェックオブジェクト

入力される単語がキーワードと合致すればそれを出力する．合致しなければ何もしない．

○サマリ作成オブジェクト

入力された合致結果を集計してサマリレポートを作成し出力する．

この3つのオブジェクトが図-6 のように部品結合されたとする．このとき同期に関する仕様として以下のことを要求する．

① キーワードも文字ストリームも有限である．(ゲート g1, g3 を通過するメッセージは有限である．)

② キーワードチェックオブジェクトでは，キーワードの入力がすべて終了してから単語切り出しオブジェクトからの入力を受け付ける．(g3 が入力終了 (eog 3) となるまで g2 を開かない．)

③ サマリレポートは必ず出力される．(g5 は必ず1回は開閉する．)

④ 最後にサマリレポートを出力する．(g5 を開いた後は halt する．)

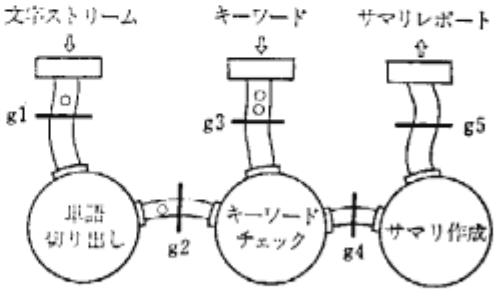

図-6　部品結合の結果

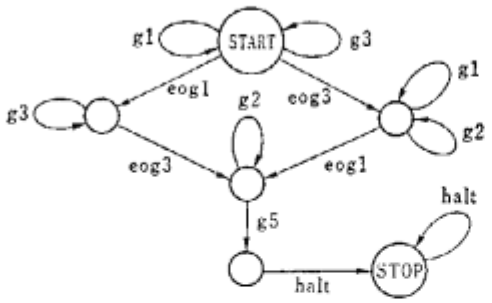

図-7 生成された状態遷移図

⑤ いつかは停止する.

これは PTL で記述すると以下になる.

① ◇eog 1∧□(eog 1⊃○(□(¬eog 1∧¬g 1)))
◇eog 3∧□(eog 3⊃○(□(¬eog 3∧¬g 3)))

② ¬g 2Ueog 3

③ ◇g 5

④ □(g 5⊃○halt)

⑤ ◇halt∧□(halt⊃○(□halt))

この仕様から図-7 に示す状態遷移図が生成される. ゲートコントローラは生成された状態遷移図に従ってゲートの開閉を行う. 言いかえれば MENDEL/87 のオブジェクトは部品に相当するが, その間のメッセージの流れを PTL で制御していることになる.

また Andler は形式的仕様として順路式 (path expression) を拡張した PPE (Predicate Path Expression) を提案し, その PPE から同期部分のプログラムを生成している[18]. PPE は正規表現を基礎とした順路式に述語を付加して変数を扱えるようにしたものである. このような述語は実行時にのみ評価される. その結果たとえば PTL では記述できない bounded buffer も扱えるようになっている. また, Dijkstra 流のセマンティクスを与えることによってたとえば Hoare などのプログラム検証手法が適用可能としている. しかし, 並行プロセスの全体としての発生する状態を陽に与えなければならないのが Wolper の手法とは大きく異なる点である.

5. おわりに

本稿では, 制御分野の自動プログラミングについて並行プロセスにおける機能部分と同期部分の合成の観点で述べた. 一般に自動プログラミングには知識処理手法[19)~21)]やプログラム変換手法[22),23)]など種々の方式があり, 今後それらの方式の制御分野への適用の可能性を十分吟味する必要がある. まず, 本稿で述べた手法に対する現状の問題点を整理すると, 次の項目があげられる.

① 問題向き言語によってその分野のユーザにとっては生産性は向上するが, 当該分野以外のユーザにとっては, 決して生産性は向上しない. 当該分野のアプリケーションが変化すると, 問題向き言語自身の変更が要求される.

② 論理式で仕様を与えて定理証明によってプログラムを合成する手法は, 論理式の記述力とその決定手続きの計算量の問題から, 制約は多い.

③ 分野を限定しない並行プロセスの機能部分および同期部分の両面を完全自動化によって支援する手法の確立には多くの課題が残されている.

これらの問題を克服するために, まず①の問題に対しては, 知識処理的アプローチが有効である. すなわち, 当該分野に関する知識を知識ベース化することにより, 非専門家にとっても問題向き言語が容易に扱えるようになる. 一方②の問題に対しては, たとえばプログラム変換法の拡張[24)]や, 論理式をプログラムとして直接実行する手法[25)]などが有用であると思われる. ③の場合においては分野を限定することなくまた完全自動化を目的とはしない手法はある. この手法には操作的アプローチがあり, その1つとして PAISLey[1)] がある. PAISLey は組込み型システムを対象としており, 並行プロセスの機能部分および同期部分についての両面について支援している. 機能部分においては functional language で記述し, 同期部分に対しては個々のプロセスを非同期に作用しあうプロセスとしてみなし, それらの関係はメッセージによる通信によって関係付けられる. PAISLey では仕様自体が実行可能になっている. この種のアプローチはそれに対する支援環境がどの程度整備されているかによってプログラムの生産性は大きく異なっている. すなわち PAISLey 以外にも SREM[26)], ENVISAGER[27)] などいわゆるソフトウェア CAD 型のシステムがあるがユーザにとって自動プログラミングとして認知されるためには環境の整備が急務であると思われる.

本稿の 4. で述べた内容は第5世代コンピュータ・プロジェクトの一環として行われた. 研究の機会を与えてくださった新世代コンピュータ技術開発機構の方々に感謝します. また, 本稿に関して有意義なコメントをいただいた(株)東芝システム・ソフトウェア技術研究所の松本一教氏に感謝します.

参 考 文 献

1) Zave, P.: An Operational Approach to Requirements Specification for Embedded Systems, IEEE Trans. Softw. Eng. Vol. SE-8, No. 3, pp. 250-269 (1982).
2) Wolper, P.: Synthesis of Communicating Processes from Temporal Logic Specification STAN-CS-82-925, Stanford Univ. (1982).
3) 松本正雄：設計とプログラミングの自動化，情報処理，Vol. 28, No. 7, pp. 862-872 (1987).
4) Gomma, H. et al.: Menu Programming—An Environment for Programming Robots, Proc. of COMPINT '85, pp. 466-470 (1985).
5) Gomma, H.: Software Development of Real-Time Systems, Comm. ACM, Vol. 29, No. 7, pp. 657-668 (1986).
6) 松本吉弘：リアルタイムシステム，昭晃堂 (1984).
7) 松本吉弘，本位田真一：プロセス制御システムへのデータベース技術の応用，情報処理，Vol. 23, No. 10, pp. 995-999 (1982).
8) Tanaka, S. et al.: New Concept Software System for Power Genaration Plant Computer Control, COPOS, Proc. of 9th PICA Conf. (1975).
9) 矢野和雄：シーケンス制御用プログラミング言語，「シーケンス制御システムにおける新しい流れ—表現法，PC の標準化，ネットワーク化を中心として—」講習会テキスト，計測自動制御学会，pp. 33-44 (1986).
10) 岸　　甫：数値制御用言語，情報処理，Vol. 22, No. 6, pp. 574-578 (1981).
11) 特集「生産におけるソフトウェア環境」，精密工学会誌，Vol. 52, No. 5, pp. 30-54 (1986).
12) Hoare, C. A. R.: Communicating Sequential Processes, Comm. ACM. Vol. 21, pp. 666-677 (1978).
13) Clarke, E. M. and Emerson, E. A.: Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic, Lecture Notes in Computer Science 131, Springer-Verlag, pp. 52-71 (1982).
14) Sistla, A. P. and Clarke, E. M.: Complexity of Propositional Linear Temporal Logics, J. ACM, Vol. 32, No. 3, pp. 733-749 (1985).
15) Moszkowski, B. C.: Reasoning about Digital Circuits, STAN-CS-83-970, Stanford Univ. (1983).
16) Uchihira, N., Kasuya, T., Matsumoto, K. and Honiden, S.: Concurrent Program Synthesis with Reusable Components using Temporal Logic, to appear in Proc. of Compsac '87 (1987).
17) 本位田真一，内平直志他：推論型システム記述言語 MENDEL，情報処理学会論文誌，Vol. 27, No. 2, pp. 219-227 (1986).
18) Andler, S.: Predicate Path Expressions, Proc. of 6th POPL, pp. 226-236 (1979).
19) Green, C.: The Design of PSI Program Synthesis System, 2nd ICSE, pp. 4-18 (1976).
20) Balzer, R., Cheatham, T. and Green, C.: Software Technology in the 1990's: Using a New Paradigm, IEEE Computer, Vol. 16, No. 11, pp. 34-45 (1983).
21) Barstow, D. et al.: An Automatic Programming System to Support an Experimental Science, 6th ICSE, pp. 360-366 (1982).
22) Burstall, R. M. and Darlington, J.: A Transformation System for Developing Recursive Programs, J. ACM, Vol. 24, No. 1, pp. 44-67 (1977).
23) Sato, T. and Tamaki, H.: Transformational Logic Program Synthesis, FGCS-84, Tokyo (1984).
24) Manna, Z. and Waldinger, R.: A Deductive Approach to Program Synthesis, ACM TOPLAS, Vol. 2, pp. 90-121 (1980).
25) Moszkowski, B. C.: Executing Temporal Logic Programs, Cambridge University Press (1986).
26) Bell, T. E. et al.: An Extendable Approach to Computer-Aided Software Requirements Engineering, IEEE Trans. S. E., Vol. SE-3, No. 1, pp. 49-60 (1977).
27) Diaz-Gonzalez, J. P. and Urban, J. E.: ENVISAGER: A Visual, Object-Oriented Specification Environment for Real-Time, Systems, Proc. of 4th International Workshop on Software Specification and Design, pp. 13-20 (1987).

(昭和 62 年 7 月 24 日受付)