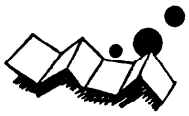
解　説

3. 応用分野の最前線



3.5 通信分野における自動プログラミング†

伊藤正樹††　市川晴久††

1. まえがき

通信システムのプログラムは，オペレーティングシステムのようなシステムプログラムの範ちゅうに属する．その通常の機能を記述することは困難ではないが，例外処理が多いこと，高い効率が要求されることが特徴であり，これらが「自動プログラミング」を困難にしている．にもかかわらず，近年通信分野においてプログラミングの自動化に対する関心が高まっている．これは，きわめて多くのユーザを抱えるシステムを対象とする場合が多いため，要求が多様化してきていることによる．また自動プログラミングの研究を促進する技術的な要因として，(1)仕様記述言語の国際的標準化が進んできていること[1)~3)]，(2)通信システム用の高級言語[4)]，オペレーティングシステム[5),6)]が確立されつつあること，があげられる．

以下では，通信ソフトウェアの特質について論じた後，通信網の処理ノードである交換機のソフトウェア，および処理ノード間の通信規約—プロトコル—に焦点をあてて最近の自動プログラミングに関する研究を概説する．特にプロトコルを重点的に扱うのは，通信システムのプログラムは単独で動作するものではなく，互いに通信し合って連携動作するものであることから，プロトコルがプログラム作成上，重要な位置を占めることによる．

2. 通信ソフトウェアの特質とモデリング

通信ソフトウェアの特質として，次に示す2種の並行性をあげることができる．

(1) 多重性．複数のユーザが互いに情報のやりとりができる状態の確立からその解除に至る過程を「呼び」といい，その呼びの制御を呼処理という．通信システムは非常に多くの呼処理を同時に行う．

(2) 各呼処理における並行性．各呼びの制御は，ユーザ端末，各種リソースなど，制御対象が本質的にもつ動作の並行性を考慮したものとなる．

この特質から通信ソフトウェアのモデル化は，多かれ少なかれ並行プロセスの概念に基づいている．各プロセスは交換プログラムの場合もプロトコルの場合も状態遷移機械としてモデル化される場合が圧倒的に多い．これは各プロセスの動作を

「メッセージ（信号）の入力 ➡ やるべき仕事の分析 ➡ 仕事（タスク）」

のサイクルとしてとらえるという考え方によっている．

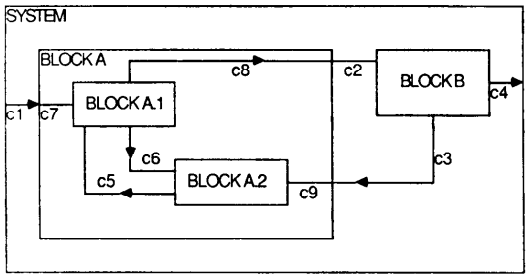
このようなモデル化に基づいた典型的な仕様記述言語として SDL (Specification and Description Language)[1)] がある．SDL では対象システムの構造をブロック (block) の集まりとして記述する (図-1)．ブロックは静的に定まったサブシステムであり，ブロックを結ぶ線はブロック間通信のためのチャネルを示す．通常プログラムに対応する部分であるプロセスは，いずれかのブロックに含まれ，状態遷移機械として記述される．

図-2 はプロセスの記述例である．状態から最初に到達するボックスはすべて信号受信を示す．その後，判断，リソースを扱う処理，信号の出力などが続き，次の状態への遷移が完了する．プロトコルの仕様記述に最も広く用いられている言語 Estelle[2)] のモジュールも類似のモデル化によっている．

排他処理を含む並行処理の記述に対し強力なペトリネット (Petri Net)[7)] は交換プログラムに適用された例は少ないがプロトコルの記述・解析にはよく用いられるモデルである[8)]．新しい傾向を示すものとして，システムを外部とのインタラクションとその順序関係のみで規定するプロトコル仕様記述言語，LOTOS[3)] がある．

† Automated Programming for Communication Control Systems by Masaki ITOH and Haruhisa ICHIKAWA (NTT Electrical Communications Laboratories).

†† NTT 電気通信研究所



ci : channel

図-1 SDL ブロック構成図の例

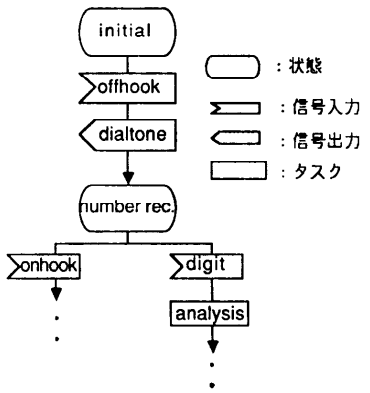


図-2 SDL/GR によるプロセス記述例

3. 交換ソフトウェア開発における自動化技術

3.1 全体的傾向

交換プログラム生成に関する研究のほとんどすべてに対していえることは，実用に供するか否かは別としてツールとして実現されていることである．逆にいえば新しい理論を打ち出しているものはごく稀で，技術的には「システム化」が中心になっている．

交換プログラムの自動生成では，ほとんどの場合，その入力が SDL により与えられる．SDL には図形表現による SDL/GR，それと等価なテキスト形式による SDL/PR，さらに各状態で獲得された各種リソースの接続関係を図示するための SDL/PE がある．現在発表されているシステムの多くは SDL/GR もしくは SDL/PR を採用している．これはプログラムイメージに近くプログラム生成が比較的容易なためと考えられる．

SDL/PE は，初期設計に適しているがプログラムとのギャップは大きい．このため，知識処理の対象となり，遷移に必要なタスク列，例外処理の自動生成に関する研究が進められている．

3.2 仕様記述言語を用いたプログラム生成

Ginsparg ら[10]は，交換機のプログラムは電話機が検知できる「ユーザからの刺激」に反応して動作する，という基本的な考え方に従った簡易仕様記述言語とそれを用いたプログラム生成法を示した．仕様記述言語は，状況 (situation, 記号 s)，ユーザの動作 (action, 記号 a)，効果 (effect, 記号 e) の3項組の集まりとして定義される．記述単位となる3項組の例を下記に示す．

s sending (message y)
a flash
e stopsend (message y), send (ak y)

意味は

message を送信中の相手 (y) がいる状況で
フッキングすると
送信を停止し，ak が y に送られる．

となる．この言語による記述はCコードに変換される．変換は原則的に "if s then e" なるプログラム a を生成するという比較的単純なものである．この手法の特徴は，自動化の重点を交換機の開発ではなく，サービスの変更・追加に置いている点であろう．

SDL からプログラムを生成するものとして Jackson らのシステム MELBA[11] を紹介する (図-3)．出力される CHILL* プログラムの最も大きい構成要素は「モジュール」であるが，MELBA ではその構を記述するために MSD (Module Structure Diagram) を提供している．これは SDL のブロック図を CHILL 用に改良したものである．プロセスの記述は SDL をそのまま用いる．MELBA への入力は，MSD の図形表現，及び SDL/GR により行われる．SDL/GR による入力は SDL/PR に変換される．これと MSD の情報，及びライブラリ情報の処理を経て CHILL コード生成に移るがこの部分は目的システム

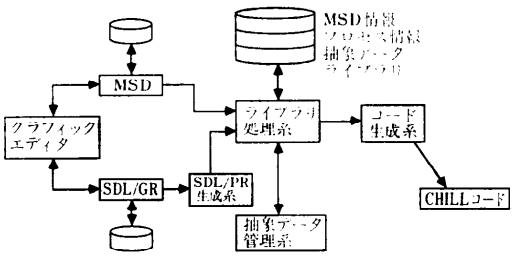


図-3 MELBA システム構成図

* Pascal 風の CCITT 標準の通信プログラム記述用の言語で CCITT HIgh Level Language の略．

に依存する部分である．MELBA ではデータ型のサポートが強化されていて，パラメータ化が可能なようにある程度抽象的に書けるようにしている．

このほか SDL を用いたプログラム生成をしめしたものとして文献12)〜16) がある．SDL/GR 以外のグラフィックインタフェースを用いたものとしては文献17)，18)がある．

著者らが開発を進めている SDE (Systems Design Environment)[19] は SDL の上位に SAL (Service Addition Language) と呼ぶサービス記述言語を設定したプログラム生成システムである．SDL がプロセスを単位とした記述であるのに対し，SAL は「サービス」を記述単位とする．サービスは各プロセスにおける関連する動作フローの組として規定され，メンテナンス段階でのサービス追加要求を，容易に記述することができる．SDL プログラムの生成は同じプロセスに属する動作フローを重ね合わせてプロセスを構成する操作を通して成される．このときユーザからサービスの選択が決定的に行われるように，プロセス間でやりとりされるメッセージが自動的に生成される．例外ケースもサービス設計者が意図した動作が明示されているため比較的特定しやすい．この検出も SDL プログラム生成時に自動的に行われる．SDL プログラムから C や CHILL のコードを出す部分では SX 1 システム[15]（最近グラフィックインタフェースが強化され Kindra と改名された）を用いている．

3.3 知識処理技術の応用

前節であげた手法における SDL 記述の対象は状態遷移に必要な処理であり，それはプログラムコードとの間に大きなギャップがないものだった．確かに交換処理は，通常動作している分にはごく単純であり，それをさらに抽象化して記述する必要性は低い．では交換プログラムに関するかぎりいわゆる AI 的な手法の適用性はないかといえば，二つの意味でその必要性が感じられる．一つは膨大な例外処理であり，もう一つはプログラムの高効率化である．

AI 技術を用いたシステムとしては津田らが開発した AIFLS 処理系[20]と堂山らの ESPA[21]がある．いずれも知識ベースシステムであり，処理対象の SDL では状態が重要な意味をもつ．これは，たとえば例外ケースを考える場合，与えられた処理に対してより状態に対してのほうがはるかに分類が容易なことによる．前に述べた SDE のように正常処理という概念が仕様記述上はっきりしていれば例外ケースの検出はできるが，「こういう状況にはこういう例外処理を挿入する」という支援機能を導入することは容易ではない．

AIFLS (Architectual Independent Formal Language for Switching software specification) は SDL 形式の言語であり，ここではユーザとシステム間のメッセージのやりとりを中心に記述される．ただし，状態に対してもステートメントを記述し，それにより処理系は各状態をリソースとその間の接続関係による記述に翻訳する．処理系は4つの知識ベースをもっていて一つめは，AIFLS の各ステートメントの内部表現への翻訳に用いられる．二つめは状態遷移規則である．三つめはハードウェアコンポーネント（リソース）の抽象的記述であり，状態の内部表現に用いられる．4つめは各リソースの状態やその接続関係を変える処理に関するものである．状態遷移規則では例外処理の挿入規則が重要である．例外ケースにはタイムアウトなどシステム内部の事情で起こるものと，ユーザが予想外の行動をしたために起こるものがあるが，それぞれ必ず対処が必要なものとそれ以外のものに分けて登録される．

プログラムの生成は次の手順で行われる．まず若干の前処理により入力された AIFLS プログラムの妥当性がチェックされた後，例外処理に対応する状態遷移の挿入がなされる．ここでは，サービスにかかわっている各ユーザ対応のプロセスが（目的とするプログラムの構造は「端末指向」となっている）それぞれどの状態で互いに同期しているかを解析したのち，例外的遷移の挿入が行われる．次に状態をリソース状態とその接続状態に翻訳し，さらに遷移に必要な処理が導出される．最終的に出力されるのはCコードである．

ESPA では，リソースなどに関する知識表現などは上と似ているがユーザインタフェースには SDL/PE を中心にしてかなり詳細な情報が表示され，専門的な情報を引き出す機能を豊富にしている．CHILL コード生成機能をもつが，SDL/PE から自動的に導出するというよりは，常にユーザに対して処理の意味を理解させながら会話的に効率的な大型交換機ソフトウェア設計を進めるためのシステムである．

4. プロトコル設計・開発の自動化

4.1 全体的傾向

計算機網に対し標準化が進んでいるプロトコルも含めたプロセス間プロトコルを対象とした研究におい

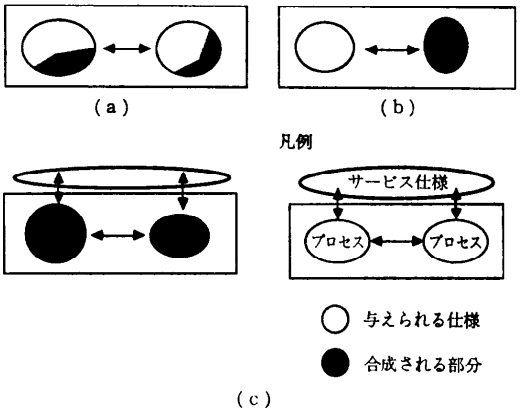


図-4　プロトコル合成のタイプ

て，プロトコルを合成したりプログラムを自動生成するための研究は仕様記述，解析・検証法に関するものに比較して，未発達といえる．プロトコルそのものを合成する手法は以下の三つが典型的である．

（1）　不完全なプロトコルを完全化する(図-4(a))．まず不完全なプロトコルを与え，次に各状態において，到着する可能性のある信号に対し，それを受理する状態遷移を付加していく手法である．

（2）　一組のプロセスのいくつかについてその動作を規定し，次に残りのプロセスの動作を自動生成する手法（図-4(b)）．

（3）　目的とするプロトコルをそのユーザ（あるいはその上位レイヤプロトコル）とのインタラクションによって規定されたプロトコル，—すなわちプロトコルのサービス仕様—に対し，その実現を与える手法(図-4(c))．

プロトコルを実現するプログラムを自動生成する手法は，十分な記述能力をもつ仕様記述言語から高級プログラム言語へのトランスレータを構成するものがほとんどである．手法の違いは使用する言語，対象システムによるところが大きい．

4.2　プロトコルの合成

前節で分類した各手法それぞれについて代表的なものを例に概説する．

[不完全なプロトコルの完全化]：Zafiropulo らによって提案された手法である[22]．図-5(a) は現在もよく用いられる Zafiropulo のプロトコルモデルの例である．プロトコルは状態遷移機械（プロセスと呼ぶ）の組として記述され，双方向に，First-in First-out のチャネルが仮定されている．各プロセスについては，丸が状態，丸を結ぶ矢印が状態遷移を示し，メッセージのやりとり（イベント，event）を示すラベルが付される．ラベルは '+' と '−' の符号をもっており，それぞれメッセージの受信，送信を表す．

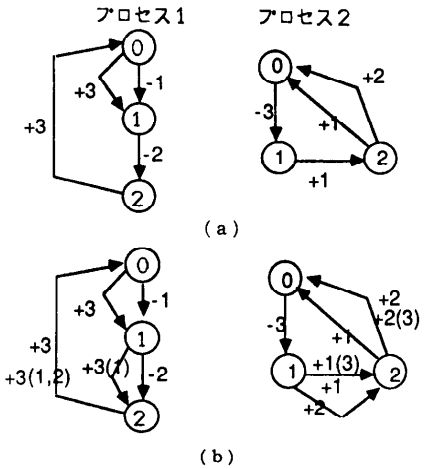


図-5　Zafiropulo のプロトコルモデルとその合成

図-5(a)に記述されたプロトコルは完全でない．たとえばプロセス2が，−3を実行した後，プロセス1が+3，−2を実行すると，どちらのプロセスもそれぞれ状態2，1から抜け出せない状況に陥る．このような可能性は，次のルールを対応する箇所すべてに適用することにより排除することができる．

ルール1　イベント '$-e$' がイベント '$-x$' に続く場合には，相手プロセスに対し次を施す．

a）すべての$+x$，$+x(s)$ に対し，$+e$，$+e(s)$ を連続させる．ここで，記号 s はメッセージの列を表し，イベント記号$+x(s)$ は，メッセージ x の送信とメッセージ列 s の送信が並行に行われ，チャネル上ですれちがった場合の相手プロセスにおけるメッセージ x の受理であることを示す．本ルールのb)，及び他のルール適用時に付加されるイベントである．

b）イベント$+x$ あるいは$+x(s)$ に連接してメッセージ送信イベントがいくつか連続しており，そのメッセージ列が s' であるとする．そのときその送信イベント列に，$+e(s')$ あるいは $+e(s,s')$ を連接させる．これを含む三つのルールを可能な限り適用すると，その結果図-5(b)に示すようにプロトコルが完備化される．なお，この操作は，適当に状態を合流させる必要があるため，プロトコル設計者と会話しながら進められる．

角田ら[23]は，上のルールをさらに増やしてアルゴリズムの効率化を図っている．また彼らは，プロセス群

のグループ分けとその「縮退」，プロセスの「分解」の概念を提案し，合成アルゴリズムを任意数のプロセスを含むプロトコルが扱えるように拡張した．

[未定義のプロセスの自動生成]：Gouda らの方法[24]では，一つのプロセスが与えられたとき，その相手として同じ形をした状態遷移図で送信・受信を反転させたものを作る．あとは互いに，送信イベントによる分枝も受信イベントによる分枝も共に可能な状態における信号のすれちがいが問題になる．この部分に関しては Zafiropulo らの手法と同様にして完備化する．ただしここではルールを強化して（人間との会話を介さず）決定的に行われる．またこの手法は必要となるバッファサイズが計算できる点を特徴とするが，その代償として扱えるプロトコルのパターンがかなり制限される．これと類似した手法として，Ramamoorthy らのもの[25]がある．

Merlin らの方法[26]は階層化プロトコルを対象にしており，着目する階層のプロトコルを，その上位階層と下位階層との間のインタラクションにより外側から規定する「サービス仕様」の概念を導入している点に価値がある．このサービス仕様と中身を構成するプロセス（モジュールと呼ばれる）のいくつかが規定された状況で，残りのプロセスを合成するというものである．入力によっては deadlock を引き起こすプロトコルを生成する場合があるなどの問題が残されている．

[サービス仕様からの合成，その他]：上に述べたサービス仕様のみから，それを実現するプロトコルを合成する手法が Bochmann らによって提案されている[27]．この方法では，まず対象とするプロトコルをそのユーザ（あるいは上位のプロトコル）との間で生起するイベントとその順序関係により規定したサービス仕様を入力としてあたえる．また各イベントには，それが生起する場所（仮に entity と呼ぶ）が記されている．プロトコルの合成は，サービス仕様を entity ごとにフィルタリングして各 entity でのイベントの順序関係を導くことで達成される．この際，当然 entity 間にわたる順序関係が問題になるがこれを満足するために entity 間の通信イベントが付加される．現状ではサービス仕様記述言語の表現能力はかなり低い．

その他の手法としては，正当性が保証された小さいプロトコルを組み合わせて大きなものを構築するものがある[28),29]．また Wolper らのアプローチが注目される[30]．これは，時相論理[31]による記述から並行プログラム言語 CSP[9] による記述を合成する試みである．時相論理は，表現能力も高くプロトコルの仕様記述への適用性も高いことから[32]今後の発展が期待される．

4.3 インプリメンテーションの自動化

Shultz らが開発した言語 FAPL (Format and Protocol Language)[33]は，PL/I をもとに階層記述，状態遷移機械の記述（行列状に記述）を導入し，データ構造の記述も SNA[33]に便利なものにチューニングされたプロトコル記述言語である．FAPL の記述は PL/I の記述に変換でき，そのプログラムはライブラリが用意された適当な環境下で動作する．

Serre らは，先に述べたプロトコル仕様記述言語 Estelle に対し，プログラム言語 Pascal へのトランスレータを作成した[34]．Estelle では並行動作単位はモジュールであり，トランスレータによりおのおの Pascal の procedure に変換される．当然複数の procedure が同時に実行可能であるが，それを管理するスケジューラは入出力処理など他の run-time サポートと同様別途用意しなければならない．モジュール対応の procedure 生成では，状態遷移条件の変換以外は複雑ではない．データ構造のサポートに関して PDU (Protocol Data Unit) の標準記述[35]との結合が期待される．Blumer らが開発したプロトコル開発システム[36]も同じ範ちゅうに属するが，検証ツールとの統合などの点でよりシステム化が進んでいる．他の仕様記述言語を用いたものには，文献 37)，38)がある．

5. 将来動向

交換ソフトウェアに関しては，各状態をリソース間の接続関係で規定し，遷移に必要な具体的手続きを書かない状態遷移記法は，プログラムをその動作ではなく「動作に対する要求，あるいは動作の意味」で規定している点でユニークである．今後リソースや，リソース間接続関係といったことがらがうまく形式的に記述できる汎用的な手法が確立されてくればさらに有効な手法となろう．特にこの記述法は，プログラミングで最も困難な部分と思われる例外処理の扱いに今後も指針を与えうる．先に述べたように現在は例外処理を知識として獲得していく方法をとっているが，「経験にたよる」域を出ていない．今後はさらに理論的な扱いが求められてこよう．

プロトコルについては，作業するプロセスやその接続関係が動的に変化するものを扱ったものはほとんどないことが指摘される．特に高位のプロトコルにおい

て，サービス開始時，終了時にはそのような変化は必ずあるし，サービスの途中で起こる場合もあろう[39]．しかもこの部分は最も例外的な現象が起きる部分でもあり，今後研究をすすめるべき部分である．

プロトコルの合成において重要視されていた「完備化」は，例外的な現象を一応カバーするという意味で有効であった．しかし，本当はどのように対処すべきかがほとんど表現されていないモデル上で，工学的な意味での収束性もあやしい処理を施すことには疑問がある．これを解決するには単純な状態遷移記述では表現能力不足であり，代数的記述[39]，論理型記述[32]などが重要視されてこよう．

6. あとがき

通信ソフトウェアにおける自動プログラミング技法を，交換機のプログラム，プロトコルを対象に概説した．プロトコルに関連した議論が長くなってしまったが，これは通信処理の並行性を顕著に代弁していると考えたからである．プロトコルに関する研究は今後も通信ソフトウェアに影響を与え続けると思う．

5. の将来動向は個人的希望も含めて書いた．今後，自動化のターゲットとして初期開発よりもメンテナンスが重要視されるようになろうが，そのためには部品，及びその間のインタフェースの「安定」が不可欠である．通信システムはその技術的成長の反映が均一ではなく，細部まで見れば多種多様な構成要素の集まりという状態は避けられない．それをうまくカバーし，各要素の「顔」が統一的にとらえられて十分管理できるような安定がほしい．

そもそも通信ソフトウェアは，「誰からもその存在を意識されないことが最も理想的な状況」という認識が一般的であろうが，ユーザ数の大きさとそれゆえシステムに対し要求される信頼性の高さを考えるとき，自動プログラミングに対する需要は他よりむしろ大きい．

参考文献

1) CCITT Recomendations Z100 to Z104.
2) ISO/TC 97/SC 21/WG 16-1 N 422 Estelle—A Formal Description Technique Based on an Extended State Transition Model.
3) ISO/TC 97/SC 21/WG 16-1 N 299 Lotos—A Formal Description Technique.
4) CCITT Recomendation Z 200.
5) Rowland, B. R. and Welsch, R. J.: The 3B20 D Processor and DMERT Operating System Software Development System, Bell Syst. Tech. J. Vol. 62, No. 1, pp. 275-289 (1983).
6) Kubota, M. and Sato, N.: Concurrent CHILL Operating System, ICC '84, 17.2 (May 1984).
7) Peterson, J. L.: Petri Net Theory and the Modeling of Systems, Prentice-Hall, Englewood Cliffs, NJ (1981).
8) Berthelot, G. and Terrat, R.: Petri Nets Theory for the Correctness of Protocols, IEEE Trans. Comput. Vol. COM-30, No. 12, pp. 2497-2505 (Dec. 1982).
9) Hoare, C. A. R.: Communicating Sequential Processes, Comm. ACM, Vol. 21, No. 8, pp. 281-302 (1978).
10) Ginsparg, J. M. and Gordon, R. D.:Automatic Programming of Communications Software via Nonprocedural Descriptions, IEEE Trans. Comput., Vol. COM-30, No. 6, pp. 1343-1347 (1982).
11) Jackson, L. N., Fidge, C. J., Pascoe, R. S. V. and Gerrand, P. H.: Computer-Aided CHILL Program Generation: Design Experience from the MELBA Project, ISS'84, S33A1 (1984).
12) Bennet, R. L., Lindner, J. A., Michelsen, R. W. and Rypka, D. J.: SDL in 5ESS Switching System Development, 6 th Int. Con. on Software Engineering for Telecommunication Switching Systems (SETSS), pp. 184-189 (1986).
13) Barra, S., Ghisio, O. and Modesti, M.: Experience and Problems of Applications of Automatic Translations from SDL Specifications into CHILL Implementations, 6th SETSS, pp. 179-183 (1986).
14) Santagostino, L., Scrignaro, D. and Serio, F.: A Computer Aided System Based on CCITT's Standards to Design and to Implement Software, 6th SETSS, pp. 101-106 (1986).
15) Dell, P. W., Jackson, L. A. and Mathews, T. J.: Computer Assistance in Support of Abstract Modelling Concepts, 5 th SETSS, pp. 19-24 (1983).
16) Bagnoli, P., Cerchio, L. and Saracco, R.: A System Design Methodology Based on SDL, 5 th SETSS, pp. 19-24 (1983).
17) Jilek, E. R.:Implementation of SDL/PR in a Digital Switching Systems, Globecom '84, S30. 1 (1984).
18) Tanabe, S., Maejima, Y., Furuya T., Tokita Y., Suzuki, T. and Shirasu, H.: Data flow Programming for Swithing System with von Neumann Machine, 6 th SETSS, pp. 18-23 (1986).
19) Ichikawa, H., Itoh, M. and Shibasaki, M.: Protocol-Oriented Service Specifications and

Their Transformation into CCITT Specification and Description Language, Trans. IECE Japan, Vol. E 69, No. 4, pp. 524-535 (Apr. 1986).
20) Hasui, K., Miyazaki, K., Shimoji, Y., Kim, M. W. and Suzuki, T.: An Intelligent Switching Software Development System—AIFLS and Its Processor, Globecom '84, S6. 4 (1984).
21) Doyama, S. and Watanabe, H.: An Expert System for ESS Software Development, 6th SETSS, pp. 118-123 (1986).
22) Zafiropulo, P., West, C. H., Rudin, H., Cowan, D. D. and Brand, D.: Towards Analyzing and Synthesizing Protocols, IEEE Trans. Comput., Vol. COM-28, No. 4, pp. 651-661 (Apr. 1980).
23) Kakuda, Y. and Wakahara, Y.: Component-Based Protocol Synthesis, Paper of Technical Group, IECE Japan, SE86-130 (1986).
24) Gouda, M. G. and Yu, Y.: Synthesis of Communicating Finite-State Machines with Guaranteed Progress, IEEE Trans. Comput., Vol. COM-32, No. 7, pp. 779-788 (July 1984).
25) Ramamoorthy, C. V., Dong, S. T. and Usuda, Y.: An Implementation of an Automated Protocol Synthesizer (APS) and Its Application to the X. 21 Protocol, ITTT Trans. Comput., Vol. SE-11, No. 9, pp. 886-908 (Sep. 1985).
26) Merlin, P. and Bochmann, G. V.: On the Construction of Submodule Specifications and Communication Protocols, TOPLAS, Vol. 5, pp. 1-25 (Jan. 1983).
27) Bochmann, G. V. and Gotzhein, R.: Deriving Protocol Specifications from Service Specifications, Proc. of the ACM SIGCOMM '86 (1986).
28) Chow, M. G., Gouda, M. G. and Lam, S. S.: On Constructing Multi-Phase Communication Protocols, Proc. 4th IFIP Int. Workshop on Protocol Specification, Testing and Verification (June 1984).
29) Choi, T. Y. and Miller, R. E.: Protocol Analysis and Synthesis by Structured Partitions, Computer Networks and ISDN Systems 11, pp. 367-381 (1986).
30) Manna, Z. and Wolper, P.: Synthesis of Communicating Processes from Temporal Logic Specifications, TOPLAS, Vol. 6, No. 1, pp. 68-93 (Jan. 1984).
31) Rescher, N. and Urquart, A.: Temporal Logic, Springer Verlag (1971).
32) Schwartz, R. L. and Melliar-Smith, P. M.: From State Machines to Temporal Logic: Specification Methods for Protocol Standards, IEEE Trans. Comput., Vol. COM-30, No. 12, pp. 2486-2496 (1982).
33) Pozefski, D. P. and Smith, F. D.: A Meta-Implementation for Systems Network Architecture, IEEE Trans. Comput., Vol. Com-30, No. 6, pp. 1348-1355 (June 1982).
34) Serre, J. M., Cenry, E. and Bochmann, G. V.: A Methodology for Implementing High-Level Communication Protocols, Proc. 19th Hawaii Int. Conf. on Systems Science (Jan. 1986).
35) CCITT Recommendation X. 409.
36) Blumer, T. P. and Sidhu, D. P.: Mechanical Verification and Automatic Implementation of Communication Protocols, IEEE Trans. Softw. Eng., Vol. SE-12, No. 8, pp. 827-843 (Aug. 1986).
37) Hansson, H. A.: Automatic Implementation of Formal Descriptions of Communication Protocols, Proc. 5th IFIP Int. Workshop on Protocol Specification, Testing and Verification (June 1984).
38) 白鳥，高橋，野口：NESDEL—プロトコル向き仕様記述言語とその応用，情報処理学会論文誌，Vol. 26, No. 6, pp. 1136-1144 (June 1985).
39) 柴崎，伊藤：ネットワークにおける多点間接続制御法の一検討，電子情報通信学会論文誌，Vol. J70, No. 2, pp. 204-211 (Feb. 1987).
40) Higashino, T., Mori, M., Sugiyama, Y., Taniguchi, K. and Kasami, T.: An Algebraic Specification of HDLC Procedures and Its Verification, IEEE Trans. Softw. Eng., Vol. SE-10, No. 6, pp. 825-836 (Nov. 1984).

(昭和 62 年3月5日受付)