マルチメディア通信と分散処理 42－3
（１９８９．５．１９）

プログラム部品利用による通信ソフトウェア開発支援

田中功一 佐藤文明 水野忠則

三菱電機(株) 情報電子研究所

通信ソフトウェアの需要は年々増大しており、その生成の自動化が望まれていたが、通信ソフトウェアには予定外の信号の例外処理が多いことや、高い効率を要求されるなど、プログラムコードの自動生成のための課題は多かった。そこで、今回、通信ソフトウェアの特徴を活かして、CCITT勧告の仕様記述言語SDLの図式表現から、C言語のプログラムコードを生成する通信ソフトウェア開発支援システムSDLプログラマを開発した。SDLプログラマは通信ソフトウェアに特化した効率的なプログラムコード生成方法、部品化支援等を設計方針として、エンジニアリングワークステーション上に実装した。SDLプログラマでは、SDLのシンボルを対象とした部品化を行なっており、通信ソフトウェアの記述に必要な、きめ細かな指定を可能として、クリティカルな処理に対しても対処可能とした。また、通信処理において欠くことのできない例外処理を容易に記述するため、部品プログラム規定時においては、C言語を拡張し、構文入れ子機能を採用した。このシステムを、実際の交換システムの一部に適用した結果、従来、人手で作成されていたプログラムコードを効率的に生成できることが分かった。

Development Support for Communication Software using Program Components

Kouichi TANAKA, Fumiaki SATO, Tadanori MIZUNO

Mitsubishi Electric Corp.
5-1-1 Ofuna, Kamakura, Kanagawa 247, Japan

In order to maintain the present rate of expansion of the communication systems, it is desirable to develop communication software efficiently by an automatic program code generation. However, the communication software has many exception processings and needs high efficiency. Therefore, an automatic program code generation has been difficult to develop for a long time. We have developed "SDL programmer" which is the communication software oriented automatic program code generation system. "SDL programmer" implemented on an engineering workstation generates C language program code from SDL/GR specification. Design plan of "SDL programmer" is consist of a communication software oriented generation and a component oriented development. In "SDL programmer", an SDL symbol is corresponding to a program component. Therefore, "SDL programmer" can be directed to generate the detailed part of processing and critical processing. We expand C language to represent a nesting structure for describing exceptional processings. Consequently, we can generate the program code of communication system efficiently by the use of this system.

1. はじめに

通信システムは、年々システムの規模が増大し、そのソフトウェアも膨大なものとなっている。そのため、通信ソフトウェアの効率的な開発法の研究が、多くの機関で行われるようになってきた。

通信システムの標準化の組織にでは、ISO(International Organization for Standardization)のEstelle[1]やLOTOS[2]、CCITT(International Telegraph and Telephone Consultative Committee)のSDL(Specification and Description Language)[3]等の仕様記述言語が規定され、それに基づく研究開発が行われている。特に、近年、仕様記述言語から直接、プログラムコードを生成する自動化に関する研究が盛んになってきているが[4-6]、通信ソフトウェアには例外処理が多いこと[4]や高い効率が要求されるなど、課題が多いのも現状である。また、ソフトウェア工学の観点からは、仕様規定からプログラム開発にいたる一連の流れを体系的に支援するとともに、プログラムの部品化と、その再利用の必要性が唱えられている[7]。

このような背景において、我々は通信ソフトウェアを効率的に開発するために、CCITT勧告のSDLを基にして、仕様作成とプログラム開発が一体化し、かつ部品合成の概念を利用したC言語プログラムコード生成システム、SDLプログラマの開発を行った。

このシステムは、人手で行っていた仕様からプログラム言語への置換の規則を、プログラミング規約として抽出した後、それを自動的に変換するシステムとして実現したものである。作成方法については、例外処理が多いといった通信ソフトウェアの特徴を活かした手法を採用した。特にここでは、この通信ソフトウェアに特化した手法を明示して、通信システムの仕様記述言語SDLの図式表現からC言語のプログラムコードを生成するシステムを報告する。

また、本システムに付随する環境としては、SDLグラフィックエディタ[8]とSDL検証ツールSDL-VWS[9]等がある。

以下、第2章では通信ソフトウェアの特徴、第3章ではSDLプログラマの設計方針と実現形態、そして第4章ではプログラムコード生成手順について述べる。さらに第5章では交換システムへの適用例を示す。

2. 通信ソフトウェアの特徴

本章では、通信ソフトウェアの特徴について述べる。通信ソフトウェアは、基本的には汎用ソフトウェアと同一の性格を有しているが、次に示す特性をもっている。

(1)状態遷移機械に基づく動作が基本である。

(2)実時間性を必要とし、特に受信処理に関しては決められた短い時間に応答する必要がある。

(3)複数のプロセスを並列的に処理しなければならないために、マルチプロセス機能が必須となる。

(4)あらゆるタイミングで発生する予定外の信号に対する処理(ここでは例外処理という)が必須となる。

上記の様な通信システムの特性に適合した仕様記述を行なうために、交換システム等を中心にSDLが用いられて来た。

SDLはCCITTが通信システムの仕様記述言語として勧告したもので、国際的にも広く使用されている。

SDLの言語仕様は、システムのブロック構造を示すブロックインタラクションダイヤグラムと、システムの動作内容を示すプロセスグラフから構成されている。また、表現方法として、意味的に等価な図式的な表現(SDL/GR)とテキスト的な表現(SDL/PR) がある。プロセスグラフは、プロセスの動きを拡張状態遷移機械のモデルの観点から、図1に示すようなシンボルを用いてグラフ的に記述するものである。

SDLで記述された仕様の例と、それに対応するC言語プログラムコードを図2及び図3に示す。図3の破線で囲まれた部分のように、例外処理の部分が備えられていることが通信ソフトウェアの特徴の一つである。

仕様書からプログラムコードへの変換は、設計者が手動で行なうのが普通であり、システムが複雑になるにつれ、コストの増大、コーディングミスの増加傾向が問題となっている。

3. SDLプログラマの設計方針と実現形態

3.1. 設計方針

SDLプログラマ(SDL programmer: Communication software generation system based on SDL)は、SDL/GR表現によるプロセスグラフで記述された仕様書から、C言語プログラムコードを生成するシステムである。 SDLプログラマを設計するにあたっては、下記の点に留意した。

(1)標準言語の使用

仕様記述言語として、SDLを用い、仕様の表現としてプロセスグラフのみを取り扱う。また、生成言語として、C言語を採用し、部品の記述には幾つかの構文を拡張した言語を用いる。

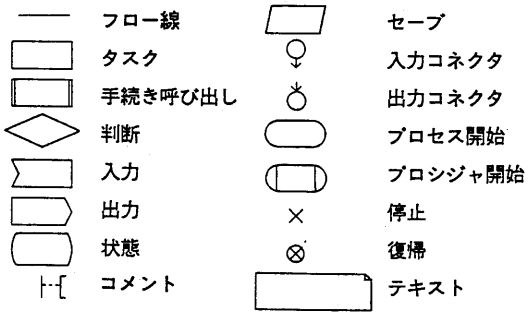


図1 SDLプログラマで用いるSDL/GRシンボル

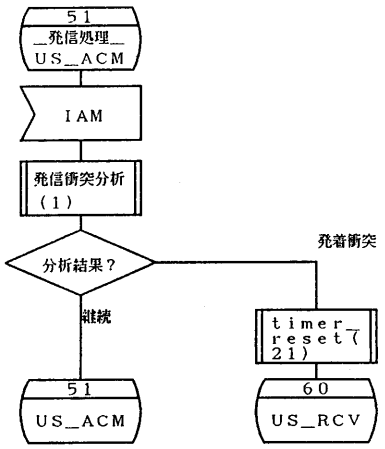


図2 SDL/GRプロセスグラフの例

```
S51: _rec(US_ACM, 0);
switch (mtypeget()) {
case M7_IAM:
        ret = dupcat(1);
        switch (ret) {
                case US_CNTN:
                        _state(state51);
                        break;
                default:
                        urtimoff(UT_T21);
                        _state(state60);
                        break;
        }
        break;
default:
        _ivdsig(0);
        _state(state51);
        break;
}
```

図3 C言語によるプログラム

(2)通信ソフトウェアに特化したプログラムコード生成

実時間性、並列性、例外処理といった通信ソフトウェアの特徴のなかで、特に生成方式にかかわる例外処理を吸収できる変換方法とする。

(3)部品化

再利用を促進し、生産効率を上げるため、プロセスグラフの状態、入力、出力、判断といったSDL/GRのシンボル一個に対して、予め部品を用意し、これに置換する。これらの部品は下位レベルのルーチンをコールする関数部と制御ステートメントで構成する。この部分を変更すると、特定の対象に対応できるようにする。関数は予めアセンブラ言語等で定義する。

(4)マンマシン・インタフェース

通信システムの設計者が出来る限りプログラム生成には手間をかけず、設計に専念するために、エラーメッセージを視覚的に表現し、ワークステーションが持つ、マウス等による優れたインタフェースを駆使して、容易に操作が行なえるツールとする。

(5)体系的支援環境

単独のツールではなく、設計からプログラムコード生成までを体系的に支援できるものとするため、SDLグラフィックエディタ[8]やSDL検証ツールSDL-VWS[9]とも統合させた体系的支援環境を構築する。

3.2. SDLプログラマを用いたプログラミングの特徴

SDLプログラマを用いるプログラミングには、以下のような特徴がある。

(1)仕様記述とプログラミングの同時実行

従来、仕様記述の後に、仕様書のレビュー、仕様の修正、プログラミング、プログラムのレビューといった流れで行われてきた作業が、仕様を中心に規定することによって、トップダウン、ボトムアップ両方のアプローチを行なうことができ、作業効率の向上が望める。また、プログラムと同内容のものを木構造チャートに入力するような従来の生成システムと異なり、処理の抽象化、視覚化が行われている。

(2)レビュー機能による仕様の確認

SDL-VWS[9]による仕様のレビュー機能で、仕様の段階で誤りや、仕様変更に対応でき、同時にプログラムに反映可能である。

(3)SDLシンタックスチェッカによる文法チェック

SDLグラフィックエディタ[8]に含まれるSDLシンタックスチェッカ及び、UNIXのLINTコマンドによって、SDL仕様の文法チェック、部品及び生成したプログラムコードの文法チェックが確実に行われる。そのため、文法誤りは、最小限に抑えられる。

(4)プロトタイピングによる効率的な開発

SDLプログラマは、一つの部品名に2種類の部品を登録できる。一つは実行用コードであり、もう一つはプロトタイプ用コードである。コード生成時に、SDLプログラマでは、実行用コードとプロトタイプ用コードを使い分けることが可能である。すなわち、プロトタイプ用の生成コードを使って、シミュレーションを行い、早期に仕様の論理的な誤りや、要求仕様との食い違いを解決できる。

(5)拡張されたC言語の構文の採用

多数の入力信号を処理する場合等で生ずる入れ子構造の位置を、 $LOGICという構文を用いて表す。これによって、プログラム中で離れて存在する例外処理等も1つの部品として容易に記述できる。

(6)コメント及び統計情報の表示

プログラムコード生成時のシンボルと部品との対応を明示するため、シンボルの内容を示すコメント情報を表示する。また、プログラム開発の管理情報を得るため、統計情報の表示を可能とする。

3.3. 実現形態

今回開発を行なったSDLプログラマは、上述の機能を持ち、かつ実使用にも十分耐えるシステムとすることを目標として、以下のように実現した。

(1)エンジニアリングワークステーション上での実現

従来、開発されてきたプログラムコード生成システムは、汎用計算機上で実行するものが多く、応答性や操作性に問題があった。我々は、32ビットアーキテクチャを有し、17インチディスプレイ、マルチウィンドウ、2ボタンマウス等の入出力装置を持つエンジニアリングワークステーションME-1000シリーズ上でSDLプログラマを開発した。

(2)プログラムコード生成対象

SDLプログラマは、通信ソフトウェアの外部仕様書もしくは内部仕様書の記述に用いられるプロセスグラフのSDL/GR表現から、実際の通信ソフトウェアの記述に使用されるC言語のプログラムコード生成を支援する。

4. プログラムコード生成手順

4.1. 生成手順

SDLプログラマは、部品プログラムフェーズ、プロセスグラフ作成フェーズ、プログラムコード生成フェーズを介して仕様書からプログラムコードの生成を支援する。

部品プログラムフェーズでは、プロセスグラフのシンボルに対する部品の内容を定義する。プロセスグラフ生成フェーズでは、内容の定義されたシンボルを部品リストから選択してプロセスグラフを作成する。プログラムコード生成フェーズでは、SDLプログラマが、プロセスグラフ生成フェーズで作成されたプロセスグラフを基に、部品プログラムフェーズで定義された部品を引用することによって、プログラムコードを生成する。基本的には、図4に示す様に部品プログラムフェーズ、プロセスグラフ作成フェーズ、プログラムコード生成フェーズの順となるが、部品の内容を後で定義可能とするため、プロセスグラフ作成フェーズを部品プログラムフェーズに先行して実行することも可能である。

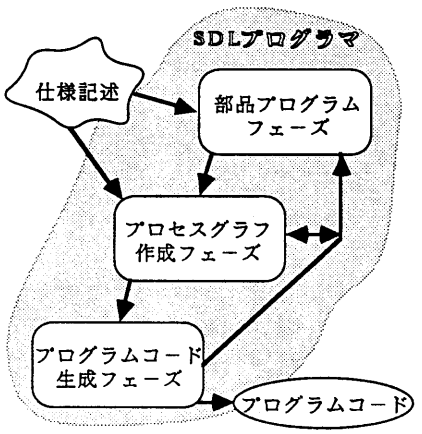


図4 プログラムコード生成手順

4.2. 部品プログラムフェーズ

部品プログラムフェーズでは、プロセスグラフのシンボルに対する部品の内容を定義する。以下に、シンボルと部品の対応付け、 $LOGIC構文を用いた入れ子構造の記述方法、そして、部品定義方法について具体的に述べる。

(1)シンボルと部品の対応

通常、一つのシンボルは一つの部品に対応する。宣言文や初期条件等は、テキストシンボルの内容として部品に登録する。なお、判断シンボルからプログラムコードを生成するた

めに、フロー線上に記述する分岐条件も、一つの部品としている。

(2)$LOGICを用いた入れ子構造の記述

図5はある状態シンボル(図5(a))とそれに対応する部品(図5(b))である。プログラムコードは図5(b)に示した拡張されたC言語を用いて記述する。図中$LOGICで示された構文はC言語に無い構文入れ子機能である。$LOGICは入れ子構造の位置を記述する構文であり、部品のプログラムコード中に使用すると、それに続く残りのシンボル(部品)のプログラムコードがこの部分に埋め込まれ、その後に、$LOGIC以下のプログラムが入る。このようにして、入れ子構造を構築することを可能とした。図6にその例を示す。この例は、図5の$LOGICで置き換えられた部分を示したものである。入れ子の制御によって、例外処理は、図中破線で囲まれた部分のように容易に記述することが可能である。

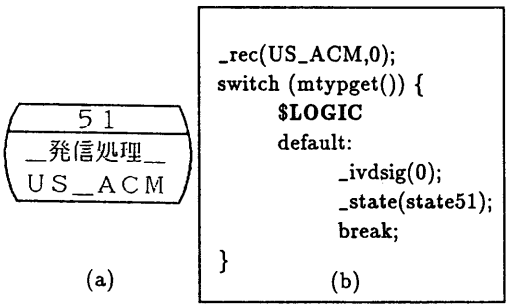


図5 シンボルとプログラムコード

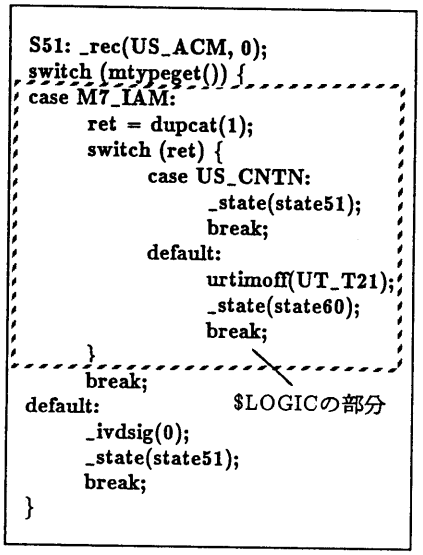


図6 $LOGICによる置き換え

(3)部品定義方法

部品の内容を編集するために、SDLプログラマは簡易なテキストエディタを備えている。また、部品を登録する際にUNIX上のLINTでシンタクスチェックを行ない、部品生成上の誤りを早い段階で見つけることを可能としている。ただし、思考の中断とならないように、チェックのみを行い、エラーが発生しても、次の操作に入ることが可能である。

4.3. プロセスグラフ作成フェーズ

プロセスグラフ作成フェーズでは、基本的に部品が定義されたシンボルを、部品リストから選択してプロセスグラフを作成する。シンボルを挿入する位置は、基本的にはSDLプログラマの有する自動誘導機能により指定する。

部品が定義されていないシンボルが必要な時は、部品プログラムフェーズに戻って部品の内容を定義するか、または部品未定義のまま、シンボルを新たに登録することができる(これを未定義シンボルという)。未定義シンボルを用いてプロセスグラフを完成してプログラムコードを生成し、あとから部品の内容を定義する事も可能である。以下に、部品リスト及びSDLプログラマの自動誘導機能について詳細に述べる。

(1)部品リスト

部品リストは、部品名、部品定義の完成度(完成もしくは未完成)、部品の引用回数、パラメータ数等から構成されている。

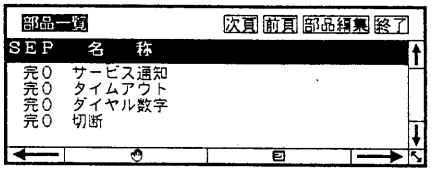
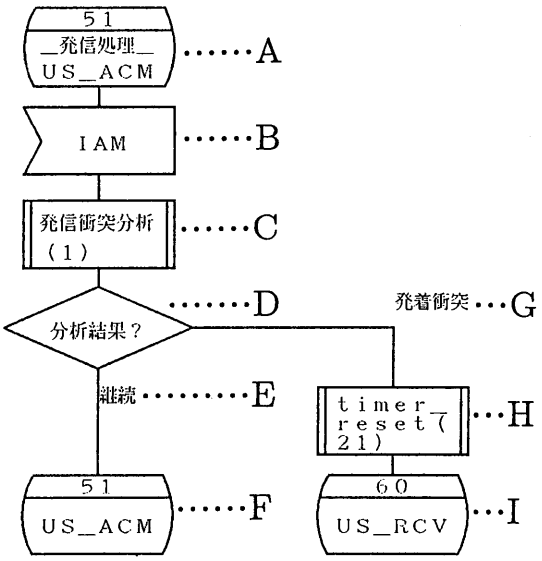


図7 部品リスト

図7の例は、入力シンボルの部品リストである。なお、部品にはパラメータを持たせて、一つの部品に対して幾種類かの動作を可能としている。パラメータを持つ部品の場合には、部品リストの表示欄P にパラメータ数が表示され、SDLプログラマでは部品選択時に自動的にパラメータの入力を促すようにしてある。設計者は、この部品リストに登録されているシンボルを組み合わせて、プロセスグラフを作成する。

(2)シンボル挿入位置の自動誘導

部品を利用してプロセスグラフを作成する場合、基本的には画面上部から下部に向かってシンボルを続けていく場合が多い。このため、SDLプログラマが自動的に次に挿入する位



(a)

```
A      /*51 : _発信処理_US_ACM*/
A S51:  _rec(US_ACM, 0);
A      switch(mtypget()) {
B            /* I AM*/
B            case M7_IAM:
C                  /*発信衝突分析(1)*/
C                  ret = dupcat(1);
D                  /*分析結果？*/
D                  switch(ret) {
E                        /*継続*/
E                        case US_CNTN:
F                              /*51 : US_ACM*/
F                              _state(state51);
F                              break;
G                        /*発着衝突*/
G                        default:
H                              /*timer_reset(21)*/
H                              utimoff(UT_T21);
I                              /*60 : US_RCV*/
I                              _state(state60);
I                              break;
D                  }
D                  break;
A            default:
A                  _ivdsig(0);
A                  _state(state51);
A                  break;
A      }
```

(b)

図8 プロセスグラフと対応するプログラムコード

置を指定する機能を設け、操作性を向上させた。直下の位置にシンボルを挿入する場合、自動的に次の挿入位置が*印で表示されるので、部品リスト中の名前をクリックすると、そこにシンボルが挿入される。*印以外の場所にシンボルを挿入する場合は、挿入位置をマウスでクリックしてから部品リストの名前をクリックする。

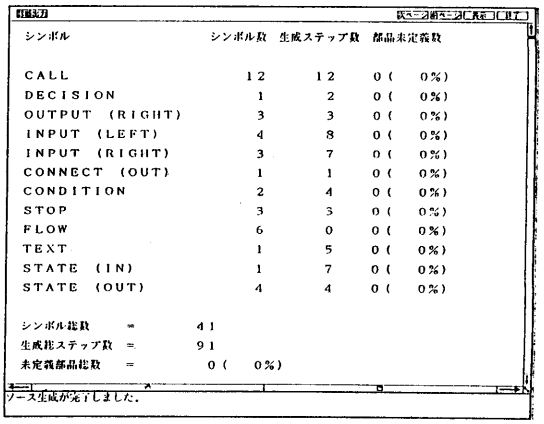
4.4. プログラムコード生成フェーズ

プログラムコード生成フェーズでは、プロセスグラフ生成フェーズで作成したプロセスグラフを基に、部品プログラムフェーズで定義した部品を引用することによって、プログラムコードを生成する。図8はプロセスグラフと、それより生成されたプログラムコードである。図8(b)のプログラムコードについているA、B、 C等の記号は、対応する図8(a)の記号が付いているシンボルから生成されたことを示す。

SDLプログラマでは生成したプログラムコードに対し、LINTを自動的に呼び出して、誤りの検出を行なう。テキストシンボルに記述された初期条件や宣言文等は、プログラムコードの先頭に付加する。また、シンボルと生成されたプログラムコードに関して、コメント情報と管理情報を表示する。

生成されたプログラムコードにおいては、元のシンボル(部品)との対応付けを明示するため、元のシンボルの内容を示すコメント情報の表示を可能にしている。表示方法は、全てのコメントを表示するモード、部分的にコメントを表示するモード、コメントを表示しないモードの3種がある。

SDLプログラマを利用した時のプログラム開発用管理情報を得るため、統計情報の表示を可能としている。管理情報の表示法には、概要モードと詳細モードの2つのタイプがある。概要モードでは図9に示すように、シンボル数、プログラムコードの生成ステップ数、未定義シンボル数、及び生成プログラムコード全体に対する割合を表示する。詳細モードでは、



シンボル	シンボル数	生成ステップ数	部品未定義数
CALL	12	12	0 (0%)
DECISION	1	2	0 (0%)
OUTPUT (RIGHT)	3	3	0 (0%)
INPUT (LEFT)	4	8	0 (0%)
INPUT (RIGHT)	3	7	0 (0%)
CONNECT (OUT)	1	1	0 (0%)
CONDITION	2	4	0 (0%)
STOP	3	3	0 (0%)
FLOW	6	0	0 (0%)
TEXT	1	5	0 (0%)
STATE (IN)	1	7	0 (0%)
STATE (OUT)	4	4	0 (0%)

シンボル総数 = 41
生成総ステップ数 = 91
未定義部品総数 = 0 (0%)

ソース生成が完了しました.

図9 管理情報の例

さらに部品の使用頻度、各シンボルのパラメータの過不足も表示する。

5. 交換システムへの適用例

SDLプログラマを実際の交換システムの呼処理に適用した。図10は、交換システムのSDLによる仕様の一部で、 SDLグラフィックエディタ[8]によって記述されたものである。

図10で右下にあるウィンドウは、部品定義用ウィンドウである。このウィンドウには状態51の部品が表示されており、イベント(信号)待ち用の関数である_recを使用している。この関数は、リターン値retにより入力信号の識別子を返すもので、この値によって各入力信号の処理へ分岐する。この部品中の$LOGICは、他の部品の継続する位置を示す。

図11には、SDLプログラマによって生成された呼処理のC言語プログラムコードを示す。このプログラムコードのコメント中には、そのシンボルの内容が記述されている。この例の部品では、生成されるプログラムコードが、一つのプロセスグラフについて一つの関数となるように定義されているので、図11のプログラムコードは、state51という一つの関数として生成されており、 LINTによってシンタックスエラーが無いことが確認されている。

6. おわりに

通信ソフトウェア向けのプログラムコード生成システムSDLプログラマの開発によって、通信システムの仕様記述言語 SDLのプロセスグラフから、C言語のプログラムコードを生成することが可能となった。SDLプログラマは、SDLグラフィックエディタを流用している編集部分も含めて、約100k

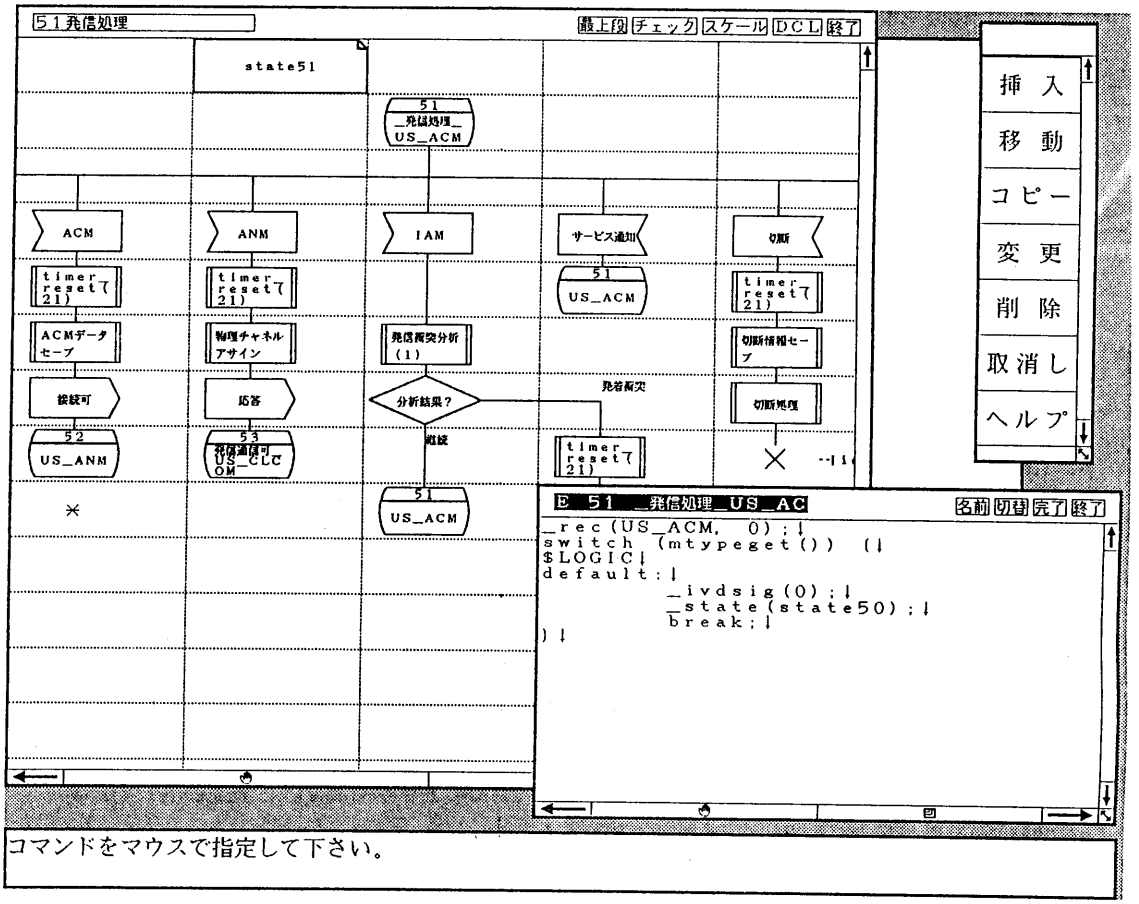


図10 交換システムへの適用例

```
/*ｓｔａｔｅ５１*/
/*発信処理ルーチン*/
#include "../include/include.h"
extern int ret, ret_code;
state51() {
    /*５１：_発信処理_US_ACM*/
S51:
    _rec(US_ACM, 0);
    switch (mtypeget()) {
    ..........
    /*ACM*/
    case M7_ACM:
        /*ｔｉｍｅｒ_ｒｅｓｅｔ（２１）*/
        urtimoff(UT_T21);
        /*ACMデータセーブ*/
        acmrec();
        /*接続可*/
        conpsbcsw();
        /*５２：US_ANM*/
        _state(state51);
        break;
    /*ANM*/
    case M7_ANM:
        /*ｔｉｍｅｒ_ｒｅｓｅｔ（２１）*/
        urtimoff(UT_T21);
        /*物理チャネルアサイン*/
        anmrec();
        /*応答*/
        anscsw();
        /*５３：発信通信可_US_CLCOM*/
        _state(state53);
        break;
    /*ＩAM*/
    case M7_IAM:
        /*発信衝突分析（１）*/
        ret = dupcat(1);
        /*分析結果？継続*/
        switch (ret) {
            case US_CNTN:
                /*５１：US_ACM*/
                _state(state51);
                break;
            /*発着衝突*/
            default:
                /*ｔｉｍｅｒ_ｒｅｓｅｔ（２１）*/
                urtimoff(UT_T21);
                /*６０：US_RCV*/
                _state(state60);
                break;
        }
        break;
        .........
    default:
        _ivdsig(0);
        _state(state51);
        break;
    }
}
```

図11 生成されたプログラムコード(一部)

ステップのソフトウェアである。

このシステムでは、$LOGICという構文を、入れ子構造の実現のために用いたが、これを用いると、それより下の階層をすべて置換することが可能である。そのため、直接シンボルに関係しない処理は先送りでき、そのシンボルに必要な処理の内容だけを記述すればよいことは、大きな利点であることが分かった。また、この構文によって、例外処理も容易に記述できることがわかった。

SDLプログラマをワークステーション上に実装した結果、ほぼ従来人手によって行なわれていた変換を代行して、効率的にプログラムコードを生成できることが分かった。現在部品の定義、プロセスグラフの作成は設計者が行っているが、今後はこれらのフェーズの効率化や自動化の度合を深めることが課題である。また、この自動生成システムは通信ソフトウェアだけでなく、一般のソフトウェアのプログラムコード生成にも適用可能と考えられる。

参考文献

[1]ISO: Information Processing Systems-Open Systems Interconnection -Estelle- A Formal Description Technique Based on an Extended State Transition Model, ISO/DIS 9074(1987).

[2]ISO: Information Processing Systems-Open Systems Interconnetion -LOTOS- A Formal Description Technique Based on the Temporal Ordering of Observational Behavior, ISO/DIS 8807 (1987).

[3]CCITT: Functional Specification and Description Language(SDL), Recommendations Z.100-104(1985).

[4]伊藤正樹、市川晴久: 通信分野における自動プログラミング, 情報処理, Vol.28, No.10, pp.1405-1411(1987).

[5]E.Vefsnmo: DASOM-A SDL-TOOL, SDL'87: State of the Art and Future Trends,R.Saracco and P.A.J.Tilanus(eds.), pp.35-42, Elservier Science Publishers B.V., North-Holland, (1987).

[6]L.N.Jackson, K.E.Cheng, T.S.Choong, R.S.V.Pascoe and G.J.Cain: MELBA at the Age of Eight: An Automatic Code Generation System, SDL'87: State of the Art and Future Trends, R.Saracco and P.A.J.Tilanus(eds.),pp.371-381,Elsevier Science Publishers B.V., North-Holland, (1987).

[7]古宮誠一、原田実: 部品合成による自動プログラミング, 情報処理, Vol.28, No.10, pp.1329-1345(1987).

[8]宗森純、水野忠則: SDLグラフィックエディタの設計と製作, 情報処理学会論文誌, Vol.29, No.7, pp.676-685(1988).

[9]宗森純、田中功一、佐藤文明、勝山光太郎、水野忠則: SDL ビジュアルウォークスルーシミュレータ, 情報処理学会研究会資料, Vol.88, No.32, ソフトウェア工学59-3(1988).

[10]ISO: OSI Basic Reference Model, ISO 7498(1984).