ソフトウェア工学 96−29
（1994. 1. 21）

通信ソフトウェア生成ツールSERIOUSの開発とOSI分散トランザクション処理への適用

辻 宏郷 † 粟屋 英司 ‡ 楠 和浩 † 中川路 哲男 †

三菱電機（株）情報システム研究所 †，コンピュータ製作所 ‡

通信ソフトウェア開発支援を目的として、形式手法に基づいた仕様化、ソースコード生成、試験等に関する、様々な研究が行われている。多くの状態や入出力イベントを持ち複雑な状態遷移を行うプロトコルにおいて、状態遷移表や形式仕様からソースコードを生成するツールの有効性は認められているものの、実際の製品開発に適用しているとの事例は少ない。我々は、従来ツールの問題点を明らかにし、実使用に必要となる要求仕様を検討した。これらの検討をふまえて、規格で定義された状態遷移表と実装依存情報を入力とし、通信ソフトウェアのプロトコルマシン部分のソースコードを生成する**通信ソフトウェア生成ツールSERIOUS**を開発した。また、OSI応用層プロトコルの一つである、OSI分散トランザクション処理に適用し、その有効性を確認した。

Communication Software Generation Tools and its application to OSI-TP Protocol

Hirosato TSUJI † Hideshi AWAYA ‡
Kazuhiro KUSUNOKI † Tetsuo NAKAKAWAJI †

Computer & Information Systems Laboratory †, Computer Works ‡

MITSUBISHI ELECTRIC CORPORATION

5-1-1 Ofuna , Kamakura, Kanagawa 247 , JAPAN

This paper discribes the application of the formal approach to the product development of the communication software implementation. At first we have discussed the issues on the former tools and the requirements for the improvement. We have developed **SERIOUS** (Software Generation Tools for OSI Upper Layer Protocols) which realize source code generation from the formal specification of the protocols and the implementation-depended information. Finally, we have applied SERIOUS to generate the protocol machine part of the OSI Distributed Transaction Processing (OSI-TP) software and confirmed the effectiveness of our methodology.

1　はじめに

通信ソフトウェア開発支援を目的として、形式手法に基づいた仕様化、ソースコード生成、試験等に関する、様々な研究が行われている。多くの状態や入出力イベントを持ち複雑な状態遷移を行うプロトコルにおいて、状態遷移表や形式仕様からソースコードを生成するツールの有効性は認められているものの、実際の製品開発に適用しているとの事例は少ない。我々は、従来ツールの問題点を明らかにし、実使用に必要となる機能仕様を検討した。これらの検討をふまえて、規格で定義された状態遷移表と実装依存情報を入力とし、通信ソフトウェアのプロトコルマシン部分のソースコードを生成する**通信ソフトウェア生成ツールSERIOUS**を開発した。そして、OSI分散トランザクション処理プロトコルに適用し、その有効性を確認した。本稿では、これらの検討結果について報告する。

2　プロトコルマシンの実現

2.1　プロトコルマシンと状態遷移表

計算機間の通信においては、通信手順を規定したプロトコルの存在が不可欠である。**プロトコルマシン**（図1）は通信ソフトウェアの中核をなす部分であり、この様なプロトコル仕様に準拠した処理を実現するメカニズムである。一般に、通信ソフトウェアにおけるプロトコルマシンは、下記に示す性質を持っている。

- プロトコルマシンは、プロトコル仕様に規定された**状態**のいずれかにある。すなわち、現在の状態を記憶する変数を持っている。
- プロトコルマシンには、論理値や数値を取り得るフラグ【注】が存在する場合がある。すなわち、これらのフラグの値を記憶する変数を持っている。
- プロトコルマシンは、外部から入力イベントを与えられることによって、入力イベント×状態の値に応じたアクションを行う。
- この時に行われるアクションは、フラグの値に応じて変化する場合がある。
- 個々のアクションにおいては、出力イベントの発行やフラグの変更などが行われる。
- プロトコルマシンは、これらのアクションの実行後、次の状態へと変化する。但し、アクションの実行のみで、状態が変化しない場合も存在する。
- 一般に、これらのプロトコル仕様は、**状態遷移表**（図2）の形態で定義されている。

本稿においては、この一連の処理を**状態遷移処理**と呼ぶ。

【注】プロトコルマシンの状態は、状態とフラグの値を合わせて意味を持つ。すなわち、状態とフラグを合わせて広義の状態とみなすことができる。一般的なプロトコル仕様の定義においては、フラグを用いることによって状態の種類を削減して、状態×入力イベント＝アクションの形式で**状態遷移表**を記述することが多い。この場合は、フラグ値に応じたアクションの変化を含めたものを、一つのアクションと定義する。

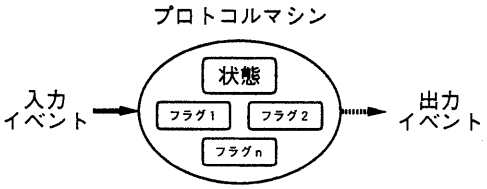

図1　プロトコルマシンのモデル

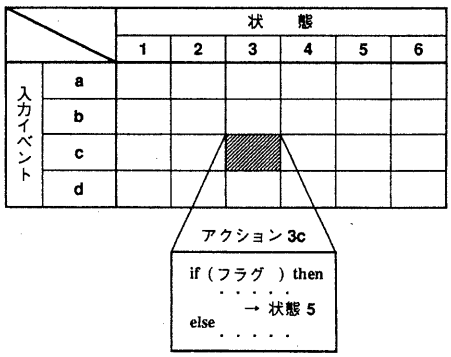

図2　状態遷移表の例

2.2 状態遷移処理オブジェクト

プロトコルマシンは、オブジェクト指向の概念を用いて、設計することができる。例えば、**オブジェクト指向言語 superC** [6] を用いる場合、次に示す様な実現方式が考えられる。

- プロトコルマシンは一つのオブジェクト（**状態遷移処理オブジェクト**）である。
- 状態やフラグは、状態遷移処理オブジェクトのインスタンス変数として表現する。
- 状態遷移処理オブジェクトは、状態遷移処理を行うためのメソッド（**状態遷移処理メソッド**）を提供する。
- 外部から入力イベントが与えられると、イベントをメソッドの引数として、状態遷移処理メソッドが呼び出される。
- 状態遷移処理メソッドでは、状態ならびに入力イベントの値に応じて、具体的な状態遷移処理を行う。

あるいは **C 言語**を用いるならば、プロトコルマシンは**関数**と**変数**の組合せによって、実現することができる。この時、状態遷移処理は、入力イベントを**関数の引数**とした**状態遷移処理関数**呼び出しで処理し、状態は、**関数内部の静的変数**によって表現する。

状態遷移処理メソッドは、状態遷移処理オブジェクトにおける処理の大半を占める部分であり、**状態×入力イベント**に応じて、対応する**アクションの実行**と**次状態への遷移**を行う。

2.3 仕様からのソースコード生成

多くの状態や入出力イベントを持つプロトコルにおいては、状態遷移表は複雑化し、これを実現する状態遷移処理のソースコード量が増大する。状態×入力イベントとアクションの対応は、状態遷移表等の計算機処理が容易な形式で与えられることから、形式手法を用いたソースコードの自動生成が適用可能である。すなわち、形式的に記述したプロトコル仕様からのプロトコルマシンを実現する状態遷移処理プログラムの生成に関して、多くの研究が報告されている [2] [3] [4] [5]。しかしながら、実際の製品開発に適用したという例は少ない。我々は、OSI 上位層通信ソフトウェア開発において、プロトコルマシンのソースコード生成ツール **SC** [8] を開発し、実システムへの適用を試みてきた。この結果、製品開発に適用する上で、下記に示す様な問題点が明らかになった。

1) 生成コードが品質管理基準をパスしない。
   - コードサイズが大き過ぎる。
   - ソース全体におけるコメントの割合等。

2) 入力ファイルの作成が容易でない。
   - 規格から入力仕様への変換が必要。
   - 結局は、実装依存情報の記述が必要。
   - 規格変更時の修正作業量が多い。

3) 利用者に拒絶されて活用されない。
   - 思い通りのコードが生成できない。
   - 多機能すぎて難しい、使いこなせない。

次章では、これらの問題点を解決するために、SERIOUS で実施した改良について述べる。

3 実システム適用のための改良

3.1 生成コードサイズの削減

状態遷移表から一定の規則に従ってソースコードを生成する場合、人手によってコーディングされたものと比較して、単純なコーディングミスは減少する反面、コードサイズが大きくなる。SERIOUS では、状態遷移表における同一処理の圧縮機能を組み込むことで、生成コードサイズを最適化した。さらに、出力ファイルの大きさによるコンパイラの制限やコードレビューの困難性を解決するために、複数ファイルに分割したソースコード出力を可能とした。

3.2 規格流用情報と実装依存情報の分離

国際標準に準拠したプロトコルを実装する場合、標準化の進展や誤りの検出によって、開発中に規格の変更が行われることがある。ツールを用いた生成では、ソースコードを直接修正することなく、状態遷移表に対応した入力ファイルの修正のみで対処可能であるが、入力ファイルには実装に依存した情報も含まれているため、実装プログラムの内部仕様に関する知識を持った者が修正を実施しなければならなかった。SERIOUSでは、入力ファイルにおける、**規格流用情報**と**実装依存情報**を明確に分離した。さらに、規格から流用する部分に関しては、状態遷移表の記述をそのまま反映可能な入力ファイル仕様とした。この結果、規格流用部分の入力・修正に関しては内部仕様に詳しくない者でも対処可能とし、規格変更に対する保守性を向上した。

3.3 適用範囲を限定した機能設計

実際の製品開発に適用するためには、実装対象プロトコルに対応した効率的なコード生成や、利用者が想定した記述スタイル通りのコード生成を行うことが要求される。SERIOUSでは、入力ファイルの記述を変更することによって、様々な形式のソースコードを生成できる様にした。しかしながら、多くの機能を盛り込み過ぎることは避け、適用範囲の絞り込みを考慮しつつ機能設計を行った。さらに、複数の選択子がある使用方法については、典型的な利用例を利用者に提示することとした。

3.4 利用者に応じた教育の実施

上記に加えて、利用者のスキルに応じた説明を行った。例えば、形式性を要求する利用者にはBNFで定義された入出力ファイル仕様を示し、数式に拒絶反応を示す様な利用者には入出力ファイルの具体例を示した。また、最初にツールの持つ全機能を公開することを避け、幾つかの機能はあえて隠しておいた。そして、習熟度の向上に従って複雑な機能を紹介していった。

4 SERIOUSの機能と構成

4.1 ツール構成

我々は、前章で述べた検討をふまえて、SERIOUS（Software Generation Tools for OSI Upper Layer Protocols）を開発した。SERIOUSは、yaccとC言語を用いて開発しており、下記の二種類のツールから構成されている。

1) **状態遷移処理メソッド生成ツール**
   一つの**状態遷移記述ファイル**と、一つ以上の**プログラム部品ファイル**を入力とし、**状態遷移処理メソッド**を生成する。
2) **アクション処理関数生成ツール**
   一つの**アクション定義ファイル**と、一つ以上の**アクション部品ファイル**を入力とし、**アクション処理関数**群を生成する。

SERIOUSの構成要素ならびに入出力ファイルの関係を、**図3**に示す。プロトコル規格で定義された**状態遷移表**に相当する部分は、それぞれのツールの入力となる、状態遷移記述ファイルとアクション定義ファイルに分割して記述する。

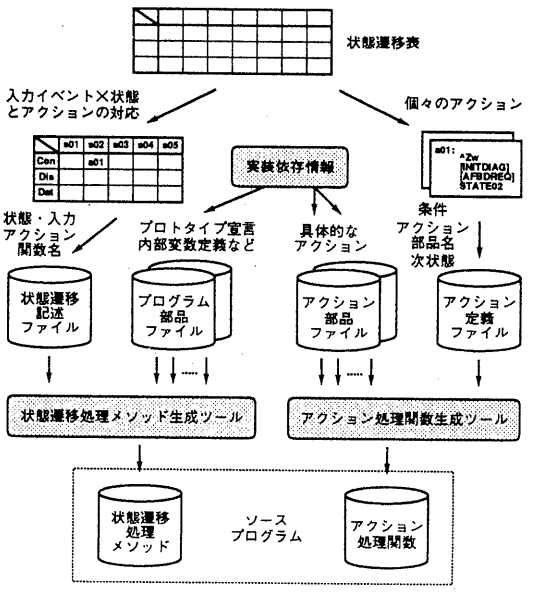

図3 SERIOUSの構成

SERIOUSを用いて生成するソースコードの言語としては、オブジェクト指向言語superCを想定している。但し、入力ファイルにおける実装依存部分の記述を変更することによって、C言語やC++のソースコードを生成することも可能となっている。

4.2 入力ファイル仕様

SERIOUSの入力ファイルは、四種類に分類することができる。下記の二種類のファイルは、実装対象プロトコルの規格で定義された**状態遷移表**に相当する部分である。

- **状態遷移記述ファイル**（例1）
  実装プロトコルの状態遷移表における「状態×入力イベント」とアクションの対応を記述するファイル。
- **アクション定義ファイル**（例2）
  状態遷移表における、個々のアクションにおける条件・アクション部品名・次状態を記述するファイル。

また、下記の二種類のファイルは、**実装依存情報**を記述する部分である。

- **プログラム部品ファイル**（例3）
  生成する状態遷移処理メソッドのプロトタイプ宣言・内部変数定義など、実装に依存した部分を記述する。
- **アクション部品ファイル**（例4）
  生成する各アクション処理関数共通のプロトタイプ宣言・内部変数定義や、個々のアクションにおける具体的なアクションの処理（イベント出力・フラグの変更など）といった実装依存部分を記述する。

4.3 生成コード仕様

SERIOUSは、状態遷移処理を実現するソースコードを、複数のファイルに分割して生成する。例えばsuperCのソースコードを生成する場合、状態遷移処理メソッドは、下記に示すファイル（メソッド・関数）に分割して生成する。

- **状態遷移処理メソッド**（例5）
  「状態×入力イベント」に応じた状態遷移

```
-------------------------------------------
-- Sample file for SERIOUS STM Generator --
-------------------------------------------
$$
/*-----------------------------------------------*/
/* Example source file "State Transition Method" */
/*   generated by SERIOUS STM Generator          */
/*-----------------------------------------------*/
$$
partsdirectory=/home/hirosato/TP/src/SERIOUS/parts
globaldef=TpGlobdef
parmdef=TpParmdef
localdef=TpLocaldef
beginaction=TpBeginAct
actionfunc=TpActFunc
defaultaction=TpDefAct
endaction=TpEndAct
eventvarible=primid
statevarible=state
statenames=MACF
actionnames=tpfm
comment
$$
event   CONNECTreq:
        state   STATE010, act110
        state   STATE020, act120
        state   STATE030, act110
event   DISCONreq:
        state   STATE020, act220
event   DATAreq:
        state   STATE015, act315
        state   STATE020, act320
        state   STATE030, act320
```

例1　状態遷移記述ファイルの例

```
-----------------------------------------
-- Sample file for SERIOUS AF Generator --
-----------------------------------------
$$
/*-------------------------------------*/
/* Example source file "Action Function" */
/*   generated by SERIOUS AF Generator   */
/*-------------------------------------*/
$$
partsdirectory=/home/hiro/TP/src/SERIOUS/parts
globaldef=AcGlobdef
funcdef=AcFuncdef
-- localdef=
-- beginaction=
endaction=AcEndAct
statevariable=state
nextstatevariable=*nstate
statenames=MACF
actionnames=tpfm
comment
$$
act120:
        (~Zw)
        [INITDIAG]
        [SETATP]
        [AFBDREQ]
                =STATE03
        (Zs)
                =STATE03
        [SETATP]
                =
act230:
        [AFEDREQ]
                =STATE00
```

例2　アクション定義ファイルの例

```
-------------------------------------------------
#include "TppmDefs.h"   /* Definitions for TPPM */

#ifndef SUPERC
extern  Uint32  state;  /* Current State Number */
extern  Flags   flags;  /* TPPM Internal Flags  */
#endif
-------------------------------------------------
Uint32  issue(tpobj)
object  *tpobj;         /* TP object from MACF */
-------------------------------------------------
Uint32  primid;         /* PrimitiveID of TP obj */
Uint32  nstate;         /* Next State No         */
Uint32  err;            /* Error Code of Action  */
-------------------------------------------------
primid = TpObject:: tpobj< get_primID >;

err = 0;        /* Clear Error Code */
-------------------------------------------------
%%(state,tpobj,&nstate,&flags,&err);
-------------------------------------------------
err = MacfStErr;        /* State Trans Err */
-------------------------------------------------
#ifdef DEBUG
fprintf(stderr,
        "MACF state : %d -> %d\n", state, nstate);
#endif

if (err==SUCCESS) {
    state = nstate;         /* Set Next State No */
    return(0);
} else {
    errlog_write(err);      /* Error Log Write   */
    state = MACFSTATE000;   /* Set Initial State */
    return(-1);
}
-------------------------------------------------
```

例３　プログラム部品ファイルの例

```
=================================================
#include "TppmDefs.h"   /* Definitions for TPPM */

extern  object  *sacf; /* SACF object */
-------------------------------------------------
Uint32  %%(state,tpobj,nstate,flags,err)
State   state;          /* Current State Number */
TpObj   tpobj;          /* TP Data Object       */
State   *nstate;        /* Next State Number    */
Flags   *flags;         /* TPPM Internal Flags  */
Uint32  *err;           /* Error Code in Action */
-------------------------------------------------
*err = SUCCESS;         /* Clear Error Code */
=================================================
$$
$$
if (SelectedFUs && TP_HANDSHAKE) {
    Zs = TRUE;
} else {
    Zs = FALSE;
}
-------------------------------------------------
$$
$$
Atppm = TRUE;   /* Set Flag "Atppm" = TRUE */
-------------------------------------------------
$$
Uint32  result; /* Result of AF-Service req */
$$
result = SacfObj:: sacf< issue, tpobj >;
         /* issue AF-BDreq to SACF */

if ( result != SUCCESS ) {
    *err = result;
    return;
}
=================================================
```

例４　アクション部品ファイルの例

（アクション処理関数呼び出し）の選択を行う。

- アクション処理関数（例６）
  個々のアクションにおける具体的なアクション処理（条件に応じたイベントの出力やフラグの変更・次状態への遷移）を行う。

アクション処理関数は、各々のアクション毎に存在する。

```
/*-----------------------------------------------*/
/* Example source file "State Transition Method" */
/*   generated by SERIOUS STM Generator          */
/*-----------------------------------------------*/

/* "TpGlobdef" */
#include "TppmDefs.h"   /* Definitions for TPPM */

#ifndef SUPERC
extern  Uint32  state;  /* Current State Number */
extern  Flags   flags;  /* TPPM Internal Flags  */
#endif

/* "TpParmdef" */
Uint32  issue(tpobj)
object  *tpobj;         /* TP object from MACF */
{
  /* "TpLocaldef" */
  Uint32  primid;         /* PrimitiveID of TP obj */
  Uint32  nstate;         /* Next State Number     */
  Uint32  err;            /* Error Code of Action  */

  /* "TpBeginAct" */
  primid = TpObject:: tpobj< get_primID >;

  err = 0;        /* Clear Error Code */

  switch(primid) {
    case CONNECTreq :
      switch(state) {
        case MACFSTATE010 :
        case MACFSTATE030 :
          tpfmact110(state,tpobj,&nstate,&flags,&err);
          break;
        case MACFSTATE020 :
          tpfmact120(state,tpobj,&nstate,&flags,&err);
          break;
        default:
          /* "TpDefAct" */
          err = MacfStErr;        /* State Trans Err */
      }
      break;

    ..........

    default:
      /* "TpDefAct" */
      err = MacfStErr;        /* State Trans Err */
  }

  /* "TpEndAct" */
#ifdef DEBUG
  fprintf(stderr,
          "MACF state : %d -> %d\n", state, nstate);
#endif

  if (err==SUCCESS) {
    ..........
  }

}
```

例５　状態遷移処理メソッドの生成例

```
/*---------------------------------------*/
/* Example source file "Action Function" */
/*   generated by SERIOUS AF Generator   */
/*---------------------------------------*/

/* "AcGlobdef" */
#include "TppmDefs.h"   /* Definitions for TPPM */

extern  object  *sacf;  /* SACF object */

Uint32  tpfmact120(state,tpobj,nstate,flags,err)
State   state;          /* Current State Number */
TpObj   tpobj;          /* TP Data Object       */
State   *nstate;        /* Next State Number    */
Flags   *flags;         /* TPPM Internal Flags  */
Uint32  *err;           /* Error Code in Action */
{
  /* Variable Part of "AFBDREQ" */
  Uint32  result; /* Result of AF-Service req */

  if (!Zw) {

    /* Behaviour Part of "INITDIAG" */
    if (SelectedFUs && TP_HANDSHAKE) {
      Zs = TRUE;
    } else {
      Zs = FALSE;
    }

    /* Behaviour Part of "SETATP" */
    Atppm = TRUE;   /* Set Flag "Atppm" = TRUE */

    /* Behaviour Part of "AFBDREQ" */
    result = SacfObj:: sacf< issue, tpobj >;
             /* issue AF-BDreq to SACF */

    if ( result != SUCCESS ) {
      *err = result;
      return;
    }

    *nstate = MACFSTATE03;

  } else {

    if (Zs) {
      ..........

    } else {
      ..........
    }
  }

  /* "AcEndAct" */
  *err = SUCCESS;          /* Clear Error Code */
}
```

例 6　アクション処理関数の生成例

5　OSI-TP への適用

5.1　OSI 分散トランザクション処理

我々は、SERIOUS の有効性を確認するために、OSI 分散トランザクション処理（OSI-TP）[1] への適用を行った。OSI-TP は、OSI 応用層規格の一つであり、状態や入出力イベントの種類が多く、複雑な状態遷移を行うプロトコルである。OSI-TP のプロトコルマシン（TPPM）は、図 4 の様にモデル化されており、複数のアソシエーションを制御する MACF、一つのアソシエーションに対応する SACF、TP APDU の処理を行う TP-ASE 等から成っている。そして、MACF と SACF 毎に状態遷移表が定義されており、合計 70 頁以上に及ぶ仕様となっている。

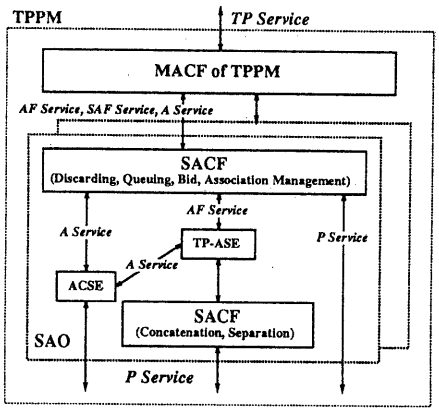

図 4　TPPM のモデル

5.2　MACF・SACF への適用結果

今回は、OSI-TP プロトコルにおいて、ダイアログ・半二重・全二重・ハンドシェークの機能単位を実現する際の MACF ならびに SACF について、状態遷移処理メソッドを SERIOUS を用いて生成した。適用結果を表 1 に示す。出力コード量は、最小（自動的なコメント文の挿入を禁止した）時の値である。

		MACF	SACF
規格	状態数	14	6
	入力イベント数	31	46
	フラグ数	33	16
入力ファイル	状態遷移記述	358 steps	283 steps
	プログラム部品	108 steps	57 steps
	アクション定義	1243 steps	539 steps
	アクション部品	506 steps	752 steps
	入力ファイル合計	2215 steps	1631 steps
	実装依存情報比率	27.7 %	49.6 %
出力	状態遷移処理メソッド	954 steps	798 steps
	アクション処理関数	8235 steps	3447 steps
	生成コード合計	9189 steps	4245 steps
入力／出力比		24.1 %	37.0 %

表 1　OSI-TP への適用結果

5.3 適用の評価と考察

前節に示した通り、生成コード量に対する入力ファイル記述量は 1/3～1/4 となった。すなわち、SERIOUS によって生成されるコードをそのまま人手で記述した場合を想定すると、実装者の作業を大幅に削減することができた。OSI 管理 (CMIP) やディレクトリの様に、状態や入出力イベント種類が少なく、単純な状態遷移のプロトコルには効果的でないが、OSI-TP の様に状態遷移表が複雑なプロトコルに対しては、SERIOUS の適用は有効であると考えられる。また、MACF と SACF への適用を比較した場合、MACF への適用がより効果的であったといえる。これは、MACF は入出力イベント・フラグの種類が共に多く、状態との組合せで発生するアクションが多岐に渡っており、仕様定義を実現する生成コード量が多くなるためである。さらに、入力ファイルにおける実装依存情報の割合も約 1/4 と少ないため、保守性も良好であると考えられる。一方 SACF では、MACF ほど複雑な状態遷移はなく、かつ入力ファイルにおける実装依存情報率が高いことから、MACF の場合の様な高い効果は得られなかった。

6 おわりに

通信ソフトウェアにおけるプロトコルマシンの状態遷移処理部分のソースコードを、規格から流用した形式仕様と実装依存情報から生成するツール SERIOUS を開発した。また、OSI 分散トランザクション処理プロトコルへの適用を行い、複雑な状態遷移を行うプロトコルに適用することの有効性を確認した。今回は、OSI 上位層プロトコルを適用対象としたが、複雑な状態遷移表を持つ他の通信プロトコルに対しても、有効であると考えられる。今後は、プロトコル変換による最適化技術などを採り入れてコード生成の効率向上を目指すと共に、他のツールとの連係による仕様検証機能の提供等について検討する予定である。

謝辞

本稿の作成にあたり有益な助言を頂いた、静岡大学工学部情報知識工学科 水野忠則教授に感謝致します。また、SERIOUS の開発に際して協力頂いた、三菱電機 (株) 情報システム研究所 田中功一氏に感謝致します。

参考文献

[1] ISO/IEC : " Information Technology - Open Systems Interconnection - Distributed Transaction Processing - Part 1~3 ", ISO/IEC 10026-1~3 (1992).

[2] 長谷川, 野村, 瀧塚 : " SDL と C を組み合わせた通信プログラム仕様の記述法及びその処理系 ", 情報処理学会研究報告 90-DPS-46-5 (1990).

[3] 野村, 長谷川, 加藤, 鈴木 : " LOTOS 実行系を用いた OSI トランザクション処理用ソフトウェアの実装と評価 ", 情報処理学会第 47 回全国大会 1G-05 (1993).

[4] 岸, 岡安, 平山, 三原 : " 状態遷移モデルに基づくプログラム部品合成システムの開発 ", 情報処理学会研究報告, 91-SE-80-22 (1991).

[5] 平山, 岡安, 深谷, 三原 : " 状態遷移モデルに基づくソフトウェア合成方式の評価 ", 情報処理学会研究報告, 92-SE-85-5 (1992).

[6] 勝山, 佐藤, 中川路, 水野 : " 通信ソフトウェア向けオブジェクト指向言語 superC ", 情報処理学会論文誌, Vol.30 No.2 (1989).

[7] Tanaka,Sato,Katsuyama,Mizuno : " AGAIN: Application protocol Gemeratpr tp accelerate implementation of network software ", The 5th SDL Forum, (1990).

[8] 田中, 辻, 佐藤, 水野 : " SC : プロトコルマシンジェネレータ ", 情報処理学会第 43 回全国大会 1P-5 (1991).