マルチメディア通信と分散処理　34－3
（１９８７・７・２４）

AdaによるOSIトランスポート/セションプロトコルの設計と実装

堀内 浩規　長谷川 亨　加藤 聰彦

国際電信電話株式会社 研究所

ISOやCCITTでは、異機種計算機･端末相互間の通信を実現するために、開放型システム間相互接続(OSI)の標準化が進められ、その実装が各方面で行われている。一方、プログラミング言語Adaは、厳密な型チェックを行うため大規模なプログラムの開発に向いており、またタスクを用いて並列動作を記述できる等の豊富な機能を有する。そこで筆者等は、AdaがOSIプロトコルを実装するためのプログラム言語の1つの候補であると考え、AdaのOSIへの適用性を検討している。そのため、複数のレーヤからなるOSI用のAdaプログラムにおいて、デッドロックやビジーウェイトを避けるタスク構成を提案し、さらにそれに基づきOSIのトランスポートプロトコルのクラス0とセションプロトコルのカーネル部分を実装した。本プログラムでは、各レーヤに対して、コネクション毎にキューの役割をするタスクとプロトコル手順を実行するタスク、及びこれらを管理するタスクを導入している。本稿ではそれらの結果について述べる。

Design and Implementation of OSI Transport / Session Protocols using Ada

Hiroki HORIUCHI　Toru HASEGAWA　Toshihiko KATO

KDD R&D Labs.　2-1-23, Nakameguro, Meguro-ku, Tokyo, 153

The standardization of OSI (Open Systems Interconnection) is promoted by ISO and CCITT and some organizations have been implementing OSI protocols. The programming language Ada has the strict type checking mechanism and many powerful features such as parallel programming using tasks. Therefore we think that Ada is a promising candidate of programming languages for the development of OSI protocols, and we have investigated the applicability of Ada to OSI. This paper proposes the OSI Ada program designing method focusing on task structure and then describes the results and considerations of the implementation of Transport protocol class 0 and Session protocol kernel according to the proposed method. In this program each layer has three types of tasks, i.e., a connection task executing protocol procedure and a queue task realizing buffering function for each connection, and the management task for each layer which manages the creations and terminations of the former two kinds of tasks.

1. はじめに

ISOやCCITTでは、異機種計算機・端末間通信を実現するために、OSIの標準化を進めており、最近では、これらOSIプロトコルを実装した製品も発表されてきている。通常、これらOSIソフトウェアの開発にあたっては、各種のプログラミング言語が採用されているが、厳密な型チェックが可能で、タスクにより並列動作を陽に記述できるAda[1]は、各階層のプロトコル・ソフトウェアを積み重ねて構成するOSI通信システムで、信頼性の高い大規模ソフトウェアを開発するための有力な言語であると考えられる。

このため筆者等は、階層構造を持ち大規模なOSIソフトウェアの特徴とAdaの言語仕様の特徴から、タスク構成及びコンパクトな状態遷移表の記述等を中心としたOSIプログラムの実装方法を提案し、これに基づきOSIトランスポート・プロトコル(TP)[2]クラス0とセション・プロトコル(SP)[3]のカーネル部分を実装した[4,5,6,7]。本稿では、提案した実装方法とTP及びSPの実装概要を述べる。

2. AdaによるOSIプログラムのタスク構成

OSIでは、7レーヤから成る参照モデルに基づき、各レーヤ毎にプロトコルが規定される。また各レーヤ間では、通常サービス定義としてインタフェースが規定される。OSIの各レーヤ、各コネクションは並列に動作するため、OSIプログラムをAdaで実装するにはAdaの並列動作の単位であるタスクを用いてどのように実装するかが重要となる。そこで、以下ではOSIプログラムのタスク構成について述べる。

2.1 基本方針

AdaによるOSIプログラムのタスク構成の検討を以下の基本方針のもとで行った。

(1) 複数のレーヤを1つのプログラムとして実現し、レーヤに対応するモジュール化はプログラム内でタスクを用いて行う。

(2) さらに各レーヤにおいて、コネクション毎のプロトコル動作を別々のタスクで実現する。このタスクは、コネクションの確立・解放に応じて動的に生成・消滅させる。

2.2 コネクションのタスクによる実現

OSIでのレーヤ間の通信は非同期式を前提としているが、Adaのタスク間の通信はエントリの呼び出しとaccept文を用いたランデブーによる同期式である。そこで、2.1の基本方針に基づき、OSIのレーヤ間の並列動作をランデブーによる同期式通信を行うAdaのタスクで行わせるため、各レーヤのコネクション用のタスク間の通信をどのように行うかについて以下の検討を行った。

先ず容易に考えられるのは、レーヤ間の通信をそのままタスク間のランデブーに対応させ、プリミティブの受信をaccept文で、プリミティブの送信をエントリ呼び出しで行う方法である。しかし、レーヤ間の通信においては送受信が独立に行われるため、下位から上位へのプリミティブの送信と上位から下位へのプリミティブの送信とがほぼ同時に生じることがあり、互いに相手のエントリを呼び出し合ってデッドロックを起こし得る(図1)。

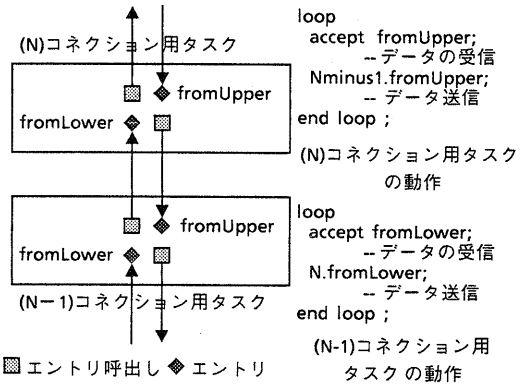

図1 デッドロックが生じる例

次に、上述のデッドロックを回避するために、コネクション用のタスクの間でバッファリングを行うキュー用のタスクを準備することを考えることができる。OSIでは、各レーヤ間に下位から上位へ及び上位から下位へプリミティブを送信するための論理的なキューが存在すると考えられ、これらのキューを別々のタスクで実現すると図2に示すようになる。

ここでは、バッファリングのためのキュー用のタスクは、隣接レーヤからのプリミティブの入出力のために、select文を用いてエントリread及びwriteが呼び出されるのを同時に待つループ構造となる。またコネクション毎のタスクは、プリミティブ受信のために隣接レーヤの2つのキュー用のタスクのエントリを同時に呼び出す必要がある。しかしこれを実現するためには、条件付エントリ呼び出しによってエントリreadを交互に呼び出す構造となり、ビジーウェイトにならざるを得ずスループットの低下が起こる。このビジーウェイトを回避するには、以下の2つの方法が考えられる。

①コネクション用のタスクを2つに分け、上位からのプリミティブの受信の処理と下位からのプリミ

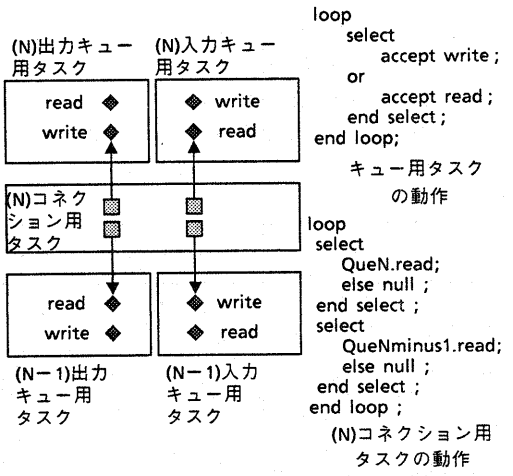

図2 ビジーウェイトが生じる例

ティブの受信の処理とを別々のタスクで実現する。

②コネクション用のタスクがプリミティブ受信のためにエントリを呼び出す2つのキュー用のタスクを1つにまとめ、上位からのプリミティブと下位からのプリミティブが1つのタスクでキューイングされる構成とする。

①②に基づく方法をそれぞれ図3、図4に示す。

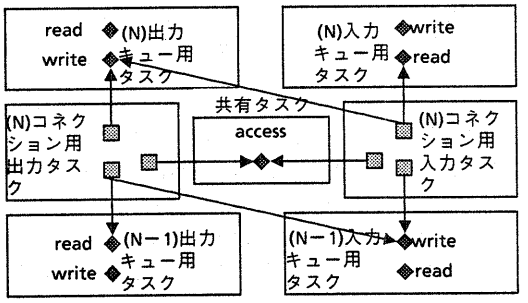

図3 ①コネクションタスクを二つに分ける方法

①の方法は送受信の並列性に着目したもので、同様な方法によりConcurrent Pascalを用いて通信プロトコルを記述した例が報告されている[8]。ここでは、1つのコネクションを2つのタスクで実現するので、状態等の情報はそれぞれのタスク間で共有する必要がある。そこで、図3に示すように、共有情報を有し排他制御を行う共有タスクを用意し、コネクション用のタスクは共有タスクのエントリaccessを呼び出して共有情報を参照更新する。

②の方法においては、コネクション毎に一つのキュー用のタスクが用意され、コネクション用タスクのプリミティブの受信はキュー用のタスクのエントリreadを呼び出して、出力は隣接レーヤのキュー用のタスクのエントリwriteを呼び出すことで実現する。

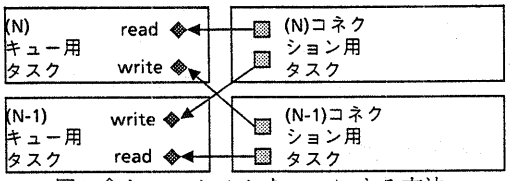

図4 ②キュータスクを一つにする方法

Adaによる実現においては、以下の点で②のタスク構成が適当と考えられる。

・各レーヤのコネクション毎のタスクの個数は①が5個、②が2個で、②の方がタスク数が少ない。

・①ではコネクション用タスクが入力プリミティブ受信の度に、共有情報を得るため共有タスクとランデブーする必要があり、スループットが低下する可能性がある。

・①では、プロトコル手順を上位からのプリミティブ受信時の処理と下位からのプリミティブ受信時の処理とに分離して抽出する必要があり、②の実現の方が容易である。

②の方法に従ったOSIプログラムのタスク構成を図5に示す。各レーヤのコネクション用のタスク(コネクション・タスク)とキュー用のタスク(キュー・タスク)は動的に生成・消滅するので、これらを一元的に管理するために、図5に示す様にレーヤ毎にマネジメント・タスクを導入している。マネジメント・タ

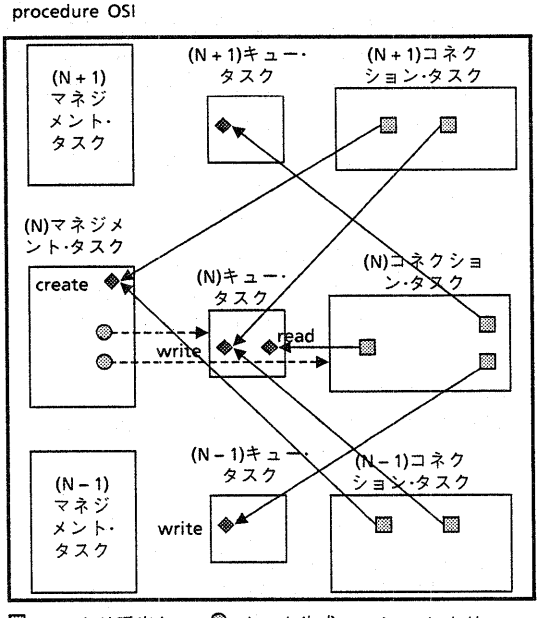

図5 AdaによるOSIプログラムのタスク構成

スクは、上位及び下位からのコネクション確立時におけるエントリcreateの呼び出しにより、コネクション・タスク及びキュー・タスクを生成する。

3. TP及びSPの実装

次に、上記の検討に基づいて、VAX 11/780(OSはVMS)上に、TPクラス0及びSPのカーネル部分のプログラム(以下TP/SPプログラムという)を実装した方法について述べる。

3.1 TP/SPプログラム実装の基本方針

本実装にあたり、以下の基本方針をたてた。

(1) 2.で提案した複数レーヤを実装するプログラム構成を、TPとSPの2レーヤの実装に当てはめる。

(2) 本プログラムはVMS上では一つのプロセスとして実現されるため、これまでC言語で開発した上位及び下位のOSIプログラム[9,10]と、VAX/VMSの提供するプロセス間インタフェースを用いて接続させる。

(3) SPの上位プログラムとTPの下位のT/Nインタフェース・プログラムとの間でやり取りされるインタフェース・プリミティブは、既存のOSIプログラム[9,10]で使用するものと同一とする。さらに、本プログラム中でTPとSPの間でやり取りされるトランスポートのインタフェース・プリミティブでも同様の構造を使用する。

(4) TPに関しては当面はクラス0の実装のみを想定するが、SPに関しては引続きSP全体の実装へ拡張できるプログラム構成とする。特にSPの状態遷移表については、SP全体に拡張した場合複雑な構造を持つため、本実装においてプログラムの作成や保守が容易な簡潔な実現を目指す。

(5) Adaのパッケージ機能を用いて保守・変更が容易なようモジュール化を行う。

以下に、2.で提案した方法に基づくTP/SPプログラムのタスク構成、状態遷移表の実現方法、パッケージ構成等について順に述べる。

3.2 タスク構成

TP/SPプログラムのタスク構成を図6に示す。この図の内で、2.で述べたマネジメント・タスク、キュー・タスク、コネクション・タスクは、TPにおいてはT_MAN/T_QUE/T_CON、SPにおいてはS_MAN/S_QUE/S_CONに対応する。

また、既に実装済みのOSIプログラム(既存システム)では、プロセス間の通信はVAX/VMSの提供するメールボックスによって行われている。そこでTP/SPプログラムでは、上位プログラム及びT/Nインタフェース・プログラムからのインタフェース・プリミティブを読み込むために、インタフェース・タスクPS_IN及びTN_INを定義している。

さらにTPまたはSPで使用されるタイマを実現するためのタイマ・タスクをコネクション・タスクの中に導入している。

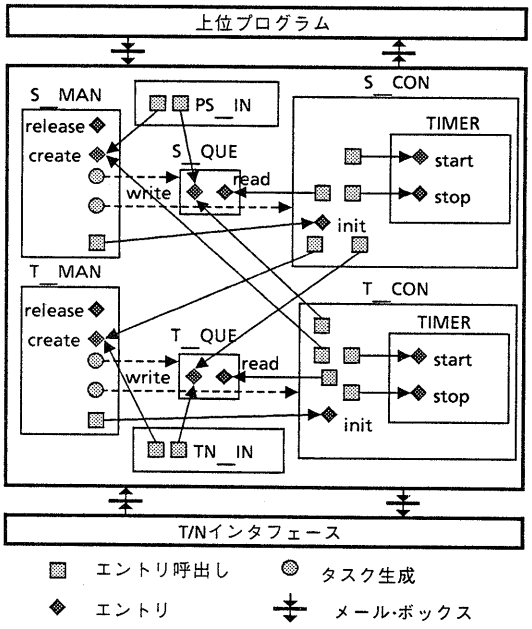

図6 プログラムのタスク構成

以下に、各タスクの機能の詳細を述べる。

3.2.1 マネジメント・タスク (S_MAN/T_MAN)

マネジメント・タスクは2.で述べたように、各レーヤに1つだけ存在し、以下の機能を実行する。

・上位からのコネクションの確立要求SCONreq/TCONreq、及び下位からのコネクションの確立指示TCONind/NCONindに対して行われるエントリcreateの呼び出しに対して、該当するレーヤのコネクション・タスク及びキュー・タスクを動的に生成する。またこのときに、TPにおけるリファレンスの決定等の、レーヤ内で一元的に行う必要のある処理を実行する。

・コネクションの解放をエントリreleaseにより検知し、対応するコネクション・タスク及びキュー・タスクをアボートにより消滅させる。

この内、コネクション・タスク及びキュー・タスクの生成はcreateのランデブーの間に行い、コネクションの確立要求/指示を発行したコネクション・タスクに、生成したキュー・タスクのアクセス値を渡す。また、生成したコネクション・タスクは自分に対応

するキュー・タスクのアクセス値を知る必要があるため、マネジメント・タスクは、createにおけるランデブーの間にコネクション・タスクのエントリinitを呼び出してこれを渡す。initの呼び出しではコネクションの確立要求/指示で運ばれるパラメータも渡される。

3.2.2 キュー・タスク (S_QUE / T_QUE)

S_QUE / T_QUEでは、それぞれセションとトランスポートのプリミティブ、トランスポートとネットワークのプリミティブをバッファリングするキューを1つ持つ。キューが空でなければ、エントリreadにおいて、対応するコネクション・タスクからの読み込み要求を受け付け、キューが一杯でなければ、エントリwriteにおいて、隣接レーヤのコネクション・タスクからの出力要求を受け付ける。

3.2.3 コネクション・タスク (S_CON / T_CON)

コネクション・タスクは、状態遷移表を管理し、プリミティブの受信に対し、遷移表に従ったプロトコル手順を実行する。

対応するキュー・タスクのエントリreadを呼び出した入力待ちの状態で、入力が与えられると、入力を解析し、状態遷移表に従って、動作の実行、プリミティブの出力、状態の遷移を行い、再び入力待ちの状態に戻る。出力は、SPからTPへ及びTPからSPへの出力については、宛先のレーヤのキュー・タスクのエントリwriteを直接呼び出す。また、既存システムへのプリミティブの出力については、コネクション・タスクがインタフェース・プリミティブを作成し直接メールボックスへ書き込む。

3.2.4 インタフェース・タスク (PS_IN / TN_IN)

既存システムでは、隣接レーヤ間でやり取りされるインタフェース・プリミティブで、Connection Identifier (CID)という識別子を用いてコネクションを識別している[9]。従ってインタフェース・タスクPS_IN / TN_INでは、次の機能が必要となる。

- ・セションとネットワークのインタフェース・プリミティブについて、受信したインタフェース・プリミティブのCIDによりコネクションを識別する。
- ・インタフェース・プリミティブが、最初のコネクション確立に関する場合はマネジメント・タスクに、その他の場合はコネクションに対応するキュー・タスクに振り分ける。そのためCIDとキュー・タスクのアクセス値の対応付けを保持する。

従って、インタフェース・タスクはコネクションの確立/解放の事象と、CIDとキュー・タスクのアクセス値の対応を知る必要がある。しかし、インタフェース・タスクは入力待ちの状態においては、VMSのシステムサービスを用いてメールボックスからの入力を待っているために、TP/SPプログラムの他のタスクからのランデブーを待つことができない。そこで今回の実装では、SCONindとNCONreqの発行時にCIDとキュー・タスクのアクセス値を含んだ生成制御情報を、またコネクション解放時には消滅制御情報を、インタフェース・プリミティブと同様なフォーマットを用いて、コネクション・タスクからメールボックス経由でインタフェース・タスクに知らせる方法をとった。

3.2.5 タイマ・タスク

TPクラス0及びSPでは、コネクション毎にTS1/TS2/TIMというタイマが実現される。タイマを実現するためには、Adaのdelay文を用いる方法、VAX/VMSの機能を用いる方法等が考えられるが、本実装ではAdaのdelay文を実行するタイマ・タスクをコネクション・タスクの子タスクとして実現する方法をとった。

タイマ・タスクは、タイマが起動される時にコネクション・タスク内に動的に生成される。生成されるとエントリstartから対応するキュータスクのアクセス値及びタイマの時間を受け取る。時間の経過内にエントリstopが呼び出されない場合は、タイムアウトの旨を対応するキュー・タスクに書き込む。タイムアウト発生時と、エントリstopが呼び出された場合には、タスクは消滅する。

3.3 状態遷移表の実現

次に、コネクション・タスクにおいてプロトコル手順を実行するための、状態遷移表をどのように実現したかについて述べる。

3.3.1 TPの状態遷移表の実現

3.1で述べたように、TP/SPプログラムでは、トランスポート・レーヤとしてTPクラス0の実装のみを想定している。TPクラス0は誤り回復機能や多重化機能等を持たず、TPの5つのクラスのうちで最も単純なクラスであり、その状態遷移表も状態数が5でイベント数が10と規模が小さい。そこで、本実装では、状態遷移表の実現には特別な工夫を行わず、次のような方法でプログラムの中に作りこん

だ。すなわち、状態と入力に対しては列挙型を定義し、コネクション・タスクは、入力が与えられると、二重のcase文によりコネクションの状態と入力の組に対して遷移が選択される動作を繰り返すループ構造とした。

3.3.2 SPの状態遷移表の実現

SPは、半二重や全二重の通信モードやデータ送受の同期/再同期等の、プロセス間の対話を管理するレーヤであり、上位プロトコルの多様な要求を充足させるため多くの機能単位を持ち大規模になっている。このため状態遷移表は、状態数が29で入力数が77と多く、さらに同一の状態と入力の組に対しても機能単位の選択やトークンの配置等により異なった動作をするという複雑な構造を持つ。勧告では、このように入力数と状態数の大きな状態遷移表を、コネクション確立状態遷移表やデータ転送状態遷移表等の8種類に分けて、正常な遷移のみを記述し、またエラーの遷移についてはまとめて扱って、記述をコンパクトにしている。

従って、SPプログラムにおいて、状態遷移表を勧告に対応させて実現すれば、開発及び保守の効率が良いと考えられる。そこでSPの状態遷移表は、プログラムの中に作りこまず陽に定数で定義し、さらに、構造体の定数の定義、名前による結合、選定子othersの利用等のAdaの定数定義機能を有効に用いて簡潔な表現とした。

以下では、図7に従って実現の詳細を述べる。まず、全状態、全入出力イベント、全動作に対して、列挙型を用いて型を定義する。これらの型では、エラーの遷移を表すために、何れにも該当しないことを示すダミーの識別子を加えている。状態遷移表毎の入力イベントは、部分範囲型を用いて全入力から括りだして定義し、また条件は整数として表現し否定は負の整数で表している。さらに、動作は1つの遷移に対して最大3個の動作に分けて記述されているので、全動作に対応する型を要素とする大きさ3の配列を用意している。そこで1つの遷移に対応する型は、条件、出力イベント、動作、次の状態を要素として持つレコード型として定義しており、エラーの遷移を表すダミーの遷移の定数も用意している。

コネクション確立状態遷移表では、1組の状態と入力に対して、条件により2つの異なった遷移を行うことがあるので、1つの遷移に対応する型を要素とする大きさ2の配列の定数として定義する。図では、入力がACで状態がST02Aである時、条件に従って2つの遷移があることを示している。また本

```
-- 全ての状態を含む列挙型
type STATE_DOM is (ST01,ST01A,…,ST21,ST22,ST713,STDUM);
-- 全ての入力イベントを含む列挙型
type IEV_DOM is (AC,CN,RF_NR,RF_R,…,SUAB_Q,TDISind);
-- 全ての動作に対応する列挙型
type ACT_DOM is (A01,A02,A03,A04, …, A30,A31,ADUM);
type ACT_ARY is array (1..3) of ACT_DOM;
-- 全ての出力イベントを含む列挙型
type OEV_DOM is (SACTD_I,SACTD_C,…,TD,SXAB_I,OEDUM);
type OEV_ARY is array (1..2) of OEV_DOM;
-- 各遷移表毎に必要な入力イベントの一部
subtype CN_DOM is IEV_DOM range AC..TCON_I;
-- 条件式に対応する型
type CND_ARY is array (1..4) of INTEGER;
-- 一つの状態遷移に対応する型
type CALIST_TYPE is record
  CND : CND_ARY := (others=>0);
  OUTEVENT : OEV_ARY := (others=>OEDUM);
  ACTION : ACT_ARY := (others=> ADUM);
  NEXTST : STATE_DOM := STDUM; end record;
-- エラーの遷移を表す定数
DUMMY_REC : constant CALIST_TYPE;
-- 一組の状態と入力に対する状態遷移に対応する型
MX_ACT : constant INTEGER := 3;
subtype ACT_NO is INTEGER range 1..MX_ACT;
type CALIST_ARY is array (ACT_NO range <>) of CALIST_TYPE;
subtype CALIST_TWO is CALIST_ARY(1..ACT_NO'FIRST+1);
-- 状態遷移表に対応する定数
type CN_TBL_TYPE is array (CN_DOM,STATE_DOM)
        of CALIST_TWO;
-- コネクション確立状態遷移表
CN_TBL: constant CN_TBL_TYPE := CN_TBL_TYPE'
(AC=>
 (ST02A=>
  (1 =>(CND =>(1=>100,others=>0),
        OUTEVNT =>(1=>SCON_CY,others=>OEDUM),
        ACTION =>(1=>A05,2=>A06,others=>ADUM),
        NEXTST =>ST713),
   2 =>(CND =>(1=>-100,2=>53,others=>0),
        OUTEVNT =>(1=>SCON_CY,2=>SPT_I),
        ACTION =>(1=>A05,2=>A06,others =>ADUM),
        NEXTST =>ST713)),…,
  others =>(others =>DUMMY_REC)),
 others =>(others =>(others =>DUMMY_REC)));
```

図7 SPの状態遷移表のプログラム表現

図に示すように、エラーの遷移は選定子othersとダミーの遷移の定数DUMMY_RECを用いてまとめて記述している。

3.4 パッケージ構成

OSIプログラムにおいても他のアプリケーションと同様に、プログラムの保守・変更が容易なようにパッケージを用いてモジュール化を行うのが適当である。本プログラムでは、図8に示す様に各レーヤ毎に状態遷移表に関係するもの、入出力のプリミティブの作成/解析の手続き群、先に述べたタスクの仕様部及び本体部をそれぞれパッケージとしてまとめ、これらのパッケージに共通して使われるデータ型等をさらに別のパッケージとしてまとめている。以下では、その概要を示す。

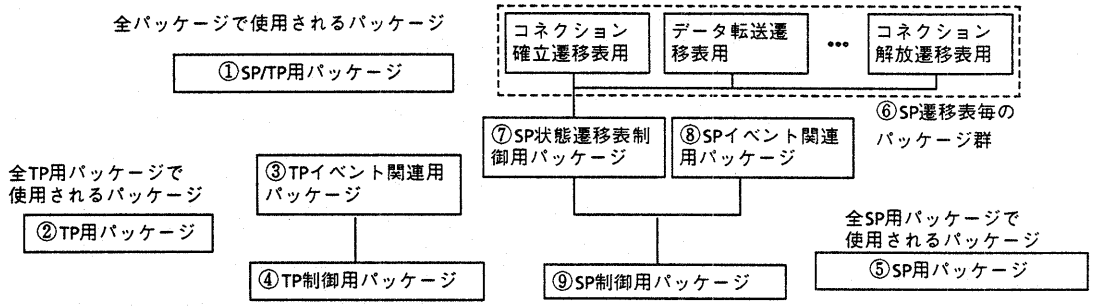

図8 SP/TPプログラムにおけるパッケージ構成

(1) ①のSP/TP用パッケージでは、両レーヤで共通に用いられるプリミティブの型等のデータ型がまとめられている。

(2) ②のTP用パッケージでは、TPの全てのパッケージで使用され、レファレンスの型定義等が含まれる。

(3) ⑤のSP用パッケージでは、3.3で述べた状態遷移表の表現に必要となるデータ型やコネクションの状態やトークンの有無等を一元的に管理を行うためのデータ型等SPのみで必要となるデータ型が含まれる。

(4) ③のTPイベント関連用パッケージ及び、⑧のSPイベント関連用パッケージでは、各レーヤで用いられる入力プリミティブ及びPDUの解析、プリミティブの種類の識別やパラメータのチェックを行う手続き、出力プリミティブ及びPDUを作成する手続き等がまとめられている。

(5) ④のTP制御用パッケージ及び、⑤のSP制御用パッケージでは、各レーヤ毎の3種のタスク及びインタフェースタスクの仕様部と本体部がまとめられている。

(6) ⑥のSP状態遷移表毎のパッケージでは、3.3で述べた状態遷移表の定数宣言が行われている。

(7) ⑦のSP状態遷移表制御用パッケージでは、状態遷移表毎のパッケージ群を使用し、それらの遷移表を操作する手続き/関数がまとめられている。具体的には、状態遷移表から入力イベントと状態の組に対応した状態遷移を取り出す手続き、条件のチェックを行う関数、動作を行う手続き等が含まれる。

3.5 システム依存機能と無検査型変換

3.5.1 システムサービスの使用

VAX/Ada[11]にはVAX/VMSが提供するシステムサービスが全てSTARLETと呼ぶパッケージに用意されており、本プログラムで必要なメールボックスの生成、入出力を行うシステムサービスもAdaプログラムで容易に使用できる。しかし、Adaプログラムが複数のタスクを含む場合には、1つのタスクの中で同期式のシステムサービスを実行すると、その終了を待っている間はプロセス自身の実行が中断されてしまい、他のタスクの動作が停止してしまう。VAX/Adaではこれを回避する同期式システムサービスをパッケージTASKING_SERVICESに用意している。本プログラムでは、インタフェース・タスク中で、メールボックスからの入力待ちを行うシステムサービスTASK_QIOWを用い、それぞれが隣接レーヤからの入力待ちの状態でも、他のタスクが動作するプログラム構成とした。

3.5.2 バイトの扱い

OSIプログラムを作成するためには、ビットまたはバイトの操作は不可欠である。Adaでは標準的にはこれらのためのデータ型は用意されていないが、VAX/Adaではシステムに依存する特性の定義を含むパッケージSYSTEMの中で、ビットの配列、バイトやワード等のデータ型を定義している。そこで本プログラムでは、プリミティブやPDUのデータ型は、この中の型を用いて記述した。

また、これらが用意されていない処理系でも内部表現節等を使い容易に定義可能であると考えられる。

3.5.3 無検査型変換

Adaは厳密な型チェックを行うことが特徴となっているが、以下の二つの部分で無検査の型変換を行う必要があった。

① SSAPアドレスとTSAPアドレス及びTSAPアドレスとNSAPアドレスの対応関係は文字列からなるファイルの中に格納した。一方プリミティブやPDUのデータ型ではこれらをバイトの配列として扱う必要があるため、ファイルから読み取ったアドレスをプリミティブやPDUに代入するために文字型とバイトの間の変換が必要であった。

② 生成制御情報及び消滅制御情報として、キュータスクのアクセス値をバイトの配列であるプリミティブとしてインタフェースタスクに渡す必要が

あり、4バイト長のアドレスであるキュータスクのアクセス値とロングワードの間の変換が必要であった。

4.結果と考察

作成されたAdaプログラムは、VAX/VMS上で既存のOSIプログラムとインタフェースし、X.25網を介して正常に動作している。以下に今回の設計及び実装で得られた結果と考察を示す。

(1) AdaによるOSIプログラムのタスク構成

OSIのレーヤ間の並列動作を、ランデブーによる同期通信を行うAdaのタスクにより実現する方法を提案し、これをTPクラス0及びSPのカーネル部分に適用した。この実装により、提案した方法はデッドロックや、ビジーウェイトによるスループットの低下が起こらないタスク構成であることが判明し、その有効性が確認された。

(2) 定数宣言を用いた状態遷移表

SPの状態遷移表の実現では、選定子othersの利用等のAdaの強力な定数定義機能によって、エラーの遷移がまとめて記述でき、状態遷移表の表現が簡潔に行えた。また、定数で定義したことによって実行時に誤って変更されることを避けることができた。

(3) パッケージによるモジュール化

パッケージを用いてモジュール化を行い、保守/変更が容易な構造とした。基本的にはレーヤ毎にパッケージを定義し、各レーヤ内部のパッケージ構成は個別に検討した。

(4) 厳密な型チェック

コンパイル時に厳密な型チェックを行うので、コンパイル時のバグの発見は容易であった。さらに、コンパイラの強力なチェック機能により、実行時に発見されたバグについても論理的な推定が可能で検出が容易であった。

(5) 汎用体

Adaの主な機能としてタスクやパッケージの他に汎用体があるが、OSIプログラムの作成においては特に必要と考えず、本実装では積極的には使用しなかった。

(6) 他言語とのインタフェース

VAX/Adaではシステム依存機能が充実しているため、他言語とのインタフェースを用いず、TPクラス0及びSPのカーネル部分を全てAdaで記述した。

(7) プログラムの規模

プログラムの実効行数は、TPクラス0が約3K行、SPのカーネル部分が約5.5K行である。実行形式の規模は262KByteであり、実行時の各タスクの記憶領域は表1に示す通りであった。但し、TPのキューの長さを64個、SPのキューの長さを32個としている。また、SPのキュータスク、SPのコネクションタスク、TPのキュータスクについては長さ節によりメモリサイズを指定する必要があった。

表1 各タスクの記憶領域の規模

タスク名	KByte
メイン	0.1
SPのマネジメントタスク	3.1
SPのキュータスク	68.7
SPのコネクションタスク	47.6
TPのマネジメントタスク	5.9
TPのキュータスク	132.0
TPのコネクションタスク	16.7
上位インタフェースタスク	2.9
下位インタフェースタスク	3.4

5. おわりに

本稿では、AdaによるOSIプログラムの設計方法、及びそれに基づいたTPクラス0及びSPのカーネル部分の実装結果を述べた。今後とも引き続きAdaによりセションプロトコル全体を実装し、その評価を行う予定である。最後に、日頃御指導頂くKDD研究所村谷所長、小野次長、浦野情報処理研究室長、鈴木主任研究員に感謝します。

参考文献

[1]: United States Department of Defence, Reference Manual for the Ada Programming Language (ANSI/MIL-STD-1815A), Jan. 1983.

[2]: CCITT, Rec. X.214, X.224, Oct. 1984.

[3]: CCITT, Rec. X.215, X.225, Oct. 1984.

[4]: 加藤,堀内,長谷川,鈴木,"AdaによるOSIトランスポート・プロトコル・クラス0の実装,"第32回情処全大,6D-6, Mar. 1986.

[5]: 堀内,長谷川,加藤,鈴木,"OSIトランスポート・プロトコルへのAdaの適用と評価,"第33回情処全大,2U-4, Oct. 1986.

[6]: 堀内,長谷川,加藤,"OSIセション層用Adaプログラムの設計,"第34回情処全大,7Z-5, Mar. 1987.

[7]: 堀内,長谷川,加藤,久木野,金子,前田,"Adaによる通信プロトコルの実現,"ソフトウェアシンポジウム'87論文集,pp227~238, June 1987.

[8]: 千葉他,"モニタを用いた端末通信機能のモデル化,"昭和56年度信学全大,1436, Apr.1986.

[9]: 鈴木,加藤,"OSIトランスポート・プロトコルのインプリメントと製品検証,"情処学会分散処理システム研究会,22-9, May 1984.

[10]: 鈴木,加藤,"OSIセッションレーヤ標準のインプリメント,"情処学会分散処理システム研究会,24-4, Nov. 1984.

[11]: Digital Equipment Corporation, VAX Ada Language Reference Manual, 1985.