マルチメディア通信と分散処理 30－3
（1986. 7. 25）

Smalltalk-80によるトランスポート・プロトコル・クラス0のインプリメント

長谷川亨　加藤聰彦　鈴木健二

国際電信電話株式会社 研究所

通信ソフトウェアの多様化・大規模化にともない、その設計の段階が重要となりつつある。特に、一般的な設計方法が決まっておらず、また評価が困難であるようなソフトウェアでは、設計において核となる部分のプログラムを実際に作成して評価する必要があると考えられる。そこで筆者らは、プログラムの生産性の高いオブジェクト指向言語Smalltalk-80を通信ソフトウェアの開発に適用することを検討しており、現在開放型システム間相互接続(OSI)のプログラムを作成している。OSIプロトコルの各レーヤは共通な構造をしているため、まず、Smalltalk-80によりOSIの各レーヤのプログラムを構成する共通の方法を考察し、その方法によりOSIトランスポート・プロトコル(TP)のクラス0を実装した。その結果、TPクラス0プログラムをソース コードが約800行のコンパクトなプログラムとして、容易に作成できる。また本プログラムは、開発済みのTPプログラムと通信実験を行うことにより正常に動作することを確認した。

Implementation of Transport Protocol Class 0 using Smalltalk-80

Toru HASEGAWA　Toshihiko KATO　Kenji SUZUKI
KDD R & D Labs.　2-1-23, Nakameguro, Meguro-ku, Tokyo, 153

The prototyping for the large and versatile telecommunication software is very important at designing stage. Especially the case where the appropriate designing principles are not fixed, the preliminary cut and try methods are useful to increase the productivity of software. We are implementing OSI protocols using the object oriented language, Smalltalk-80, so as to investigate its applicability to the prototyping. In the OSI system the structure of each layer is similar and therefore we present the general method for implementing all layer programs by Smalltalk-80, and then have implemented the OSI Transport Protocol class 0 using this method. This program is very compact, whose size is about 800 steps, and shows the applicability to the development of OSI software. This program can communicate successfully with the OSI system which implemented Transport Protocol class 0.

1. はじめに

通信ソフトウェアの多様化･大規模化にともない、その設計の段階が重要となりつつある。特に、一般的な設計方法が決まっておらず、また評価が困難であるようなソフトウェアでは、設計において核となる部分のプログラムを実際に作成して評価する必要があると考えられる。一方、プログラムの生産性を高めることを目的とした言語としてオブジェクト指向言語Smalltalk-80[1]が提案されている。

そこで、筆者らは通信ソフトウェアのプロトタイピングにオブジェクト指向言語Smalltalk-80を用いることを検討しており、現在開放型システム間相互接続(OSI)のプログラムを作成している[2]。本稿では、Smalltalk-80でのOSIプログラムの構成方法を提案し、その方法によりOSIトランスポートプロトコル(TP)[3]のクラス0を実装したので、その結果と問題点について報告する。

2. Smalltalk-80を用いたOSIプログラムの設計方針

2.1 Smalltalk-80の特徴

Smalltalk-80において、操作される対象は全てオブジェクトとして表現され、オブジェクトの振る舞いは、クラスとしてまとめて抽象的に記述される。実際には、オブジェクトはクラスのインスタンスとして実現され、その実行はオブジェクトが他のオブジェクトからのメッセージを受け取り、それに対応する実行の手続きとしてのメソッドを実行することにより行われる。クラスではそのオブジェクトが行う動作の全てが、メソッドとして定義される。また、オブジェクトはその内部状態を示すためにインスタンス変数を持つ。

クラスには継承の機能があり、システム標準か、または既に作成済みのクラスを、親クラスとして新しいクラスを作成できる。この場合、新しいクラスでは親クラスで定義されているメソッドを利用でき新たに作成する必要はない。Smalltalk-80では、この継承の機能を用いた差分プログラミングが重要な特徴になっている。

2.2 設計方針

今回のTPクラス0の作成では、引き続きOSIの各レーヤの実装に適用できるように、一般性の高いプログラムの作成を目的として、以下の方針を立てた。

①OSIの全レーヤは共通な構成を持っており、Smalltalk-80により全レーヤを共通に構成するために、いかなるプログラム構造を持てばよいかを検討し、その結果をTPクラス0の作成に適用する。

②各レーヤにおいて共通な要素をSmalltalk-80におけるクラスとして定義し、他のレーヤのプログラムが再利用可能にする。

本実装では、主に①に基づいて実装を行い、②についても検討を行った。

3. Smalltalk-80を用いたOSIプログラムの構成方法

3.1 OSIプログラムの一般的構成

OSIプログラムは一般に、複数のレーヤより構成され、各レーヤは隣接するレーヤとキューを介してインタフェース･プリミティブをやりとりし合いながら、並列に動作すると考えられる。さらに各レーヤの機能は、コネクション毎に有限状態機械としてプロトコル手順を実行する機能と、TPのレファレンス等のレーヤ内で一元的に割り当てられる情報を保持し、コネクションの確立･解放に応じてコネクションを管理する機能から構成される。

そこでSmalltalk-80によりOSIプログラムを作成するには、上記のような各レーヤの構成要素をオブジェクトとして記述する必要があり、次にOSIプログラムのオブジェクト構成について述べる。

3.2 レーヤのオブジェクト構成

各レーヤに対して、上述のコネクション毎の動作、コネクション管理の機能及びコネクションエンドポイント(CEP)に対応するレーヤ間のキューをオブジェクトに対応付け、これらをそれぞれコネクション対応オブジェクト、管理用オブジェクト、及びキューオブジェクトと呼ぶ。

コネクション対応オブジェクトとキューオブジェクトは、コネクションの確立･解放にともない生成･消滅する。また、管理用オブジェクトは常に唯一つ存在し、レーヤ内で一元的に割り当てられる情報を管理する。また、コネクション対応オブジェクトとキューオブジェクトの生成は、管理用オブジェクトにより行われ、消滅する場合は管理用オブジェクトに通知される。

管理用オブジェクトは、接続要求に対して生成したキューのポインタを要求元に戻し、また接続要求及び接続指示を一つずつ処理する必要がある。従って、インタフェース･プリミティブの内、接続要求と接続指示はキューを介してやりとりされる必要はなく、管理用オブジェクトへのメッセージにより実現される。それ以外のインタフェース･プリミティブは隣接するコネクション対応オブジェクト間でキューオブジェクトを介してやりとりされる。ここでは、キューへの書き込み時には書き込み要求

の引数となり、キューからの読み出し時には読み出し要求の戻り値になる。

これらのオブジェクトの間の関係をTPを例にとり図1に示す。

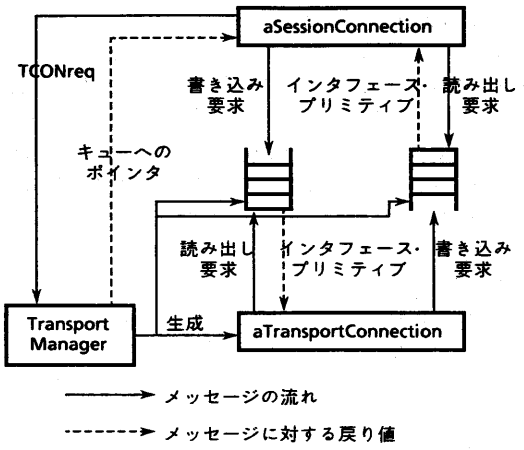

図1 オブジェクトの間の関係

この図に示すように上位のコネクション対応オブジェクト(aSessionConnection)は、下位のコネクションを確立するために接続要求のメッセージ(TCONreq)を管理用オブジェクト(TransportManager)に送る。TransportManagerはコネクション対応オブジェクトとキューオブジェクトを生成し、キューオブジェクトのポインタをTCONreqの戻り値としてaSessionConnectionに通知する。その後は、直接キューオブジェクトを介してインタフェース・プリミティブをやりとりする。

3.3 Smalltalk-80による並列動作の実現方法

3.2で述べた構成のOSIプログラムをSmalltalk-80により実現するためには、特にコネクション対応オブジェクトをいかに並列に実行させるかが問題となる。ここでは、Smalltalk-80の並列動作記述機能を用いて、OSIプログラムを構成する方法について述べる。

3.3.1 Smalltalk-80での並列動作記述

Smalltalk-80では並列動作を記述するためにProcessとProcessorSchedulerの2つのクラスが定義されている[4]。並列に実行させる一連の処理はProcessのインスタンス(プロセスと呼ばれる)として実現され、プロセスは一連の動作をまとめたブロックにメッセージnewProcessを送ることにより生成される。プロセスのスケジューリングはProcessorSchedulerのインスタンスProcessorにより行われ、実行するプロセスは優先度の高い順に、同一の優先度の場合はラウンド・ロビン方式で選択される。但し、プロセス切り替えは、実行されているプロセス自身で実行を一時中断するか、より優先度の高いプロセスがスケジュールされない限り行われない。従って、実際に複数のプロセスを切り替えながら並列に動作させるには、実行されているプロセスの中でProcessorに、実行待ち行列内のプロセスに実行権を譲ることを要求するメッセージyieldを送る必要がある。

また並列に動作するプロセス間の同期や情報のやり取りのためにSemaphoreとSharedQueueの2つのクラスが定義されている。

プロセス間で情報を共有する場合、プロセスの相互排除を行うためにクラスSemaphoreを用いる。まずSemaphoreにメッセージforMutualExclusionを送ることによりセマフォを生成する。次に、相互排除すべき動作をブロックで表し、メッセージcriticalの引数としてそのセマフォに送ることにより、相互排除が実現できる。メッセージcriticalでは、まずセマフォに対して共有する情報が解放されるまで待つためのメッセージwaitを送り、解放された後そのブロックを実行し、終了後に情報の解放を行うメッセージsingalをセマフォに送る。

プロセス間で同期を取りながら情報をやりとりするのに必要なキューを実現するために、クラスSharedQueueを用いる。SharedQueueにより実現されるキューに対しては、メッセージnextにより読み出し、メッセージnextPutにより書き込みを行うことができる。クラスSharedQueueは、クラスSemaphoreの機能を用いている。

3.3.2 OSIプログラムでの並列動作の実現

3.2で述べた構成のOSIプログラムでの一連の処理は、コネクション対応オブジェクトが、他レーヤからのインタフェースプリミティブをキューから読み取り、有限状態機械に基づいた処理を行い、出力のインタフェースプリミティブを目的のレーヤのキューに書き込むまでの動作である。従って、OSIプログラムにおける並列動作は、上記の一連の処理を単位として並列に動作させることにより実現できる。

従って、コネクション対応オブジェクトでは、一つのキューからのインタフェースプリミティブに対する一連の処理にプロセスを割当てることとする。また、プロセス切り替えは、簡単にするため

に、一連の処理が終了する度にProcessorにメッセージyieldを送ることにより行うこととする。

ここで、コネクション対応オブジェクトでは、上位及び下位からのインタフェースプリミティブに対する処理が別のプロセスとして並列に動作するので、一つのコネクションの状態を複数のプロセスで共有することになる。そこで、有限状態機械に基づく処理を非可分するために、すなわち、2つのプロセスが同時にこの処理を実行しないように、コネクション対応オブジェクトの有限状態機械に基づく処理を相互排除する必要がある。

コネクション対応オブジェクト間のキューは、並列動作を行うプロセス間の同期を取るために、クラスSharedQueueのインスタンスとする。

一方、コネクションの確立･解放が行われる時には、レーヤ内で一元的に管理される情報が書き換えられるので、同時にコネクションの確立･解放を行ってはいけない。従って、管理用オブジェクトでの処理も相互排除する必要がある。

また、キューにインタフェース･プリミティブが無い場合、プロセスはキューに隣接するレーヤからインタフェース･プリミティブが書き込まれるまでサスペンドし、どのプロセスもビジー･ウェイトにならない構成とした。

4. TPクラス0のSmalltalk-80での実現

4.1 概要

TPクラス0プログラムに対して、3章の管理用オブジェクトとコネクション対応オブジェクトを、それぞれ新しいクラス(TransportManagerとTransportConection)として定義した。またキューオブジェクトは、コネクションエンドポイント(CEP)に対応するので、隣接するレーヤ間の上下2方向のキューをまとめて、新しいクラスCepQueueとして定義した。それ以外のオブジェクトでは、Smalltalk-80で標準的に提供するクラスでは記述しにくいものだけを、新たなクラスとして定義した。TPクラス0プログラムのために作成したクラスは7個である。

TPクラス0プログラムにより実際に通信を行うためには、さらにネットワークレーヤのプログラムと、セションレーヤに位置するユーザインタフェースのプログラムが必要であり、それらも併せて作成した。但し、ネットワークレーヤにはRS232Cの回線を用いて通信させるための簡易な手順を用いた。

以下に、TPクラス0のために作成した主なクラス及び既存のクラスを用いて実現した主な機能について述べ、さらにネットワークレーヤ及びユーザインタフェースのプログラムについても述べる。但し、本実装では、新しいクラスを定義する時に、クラスの継承機能を積極的には利用しなかつたが、共

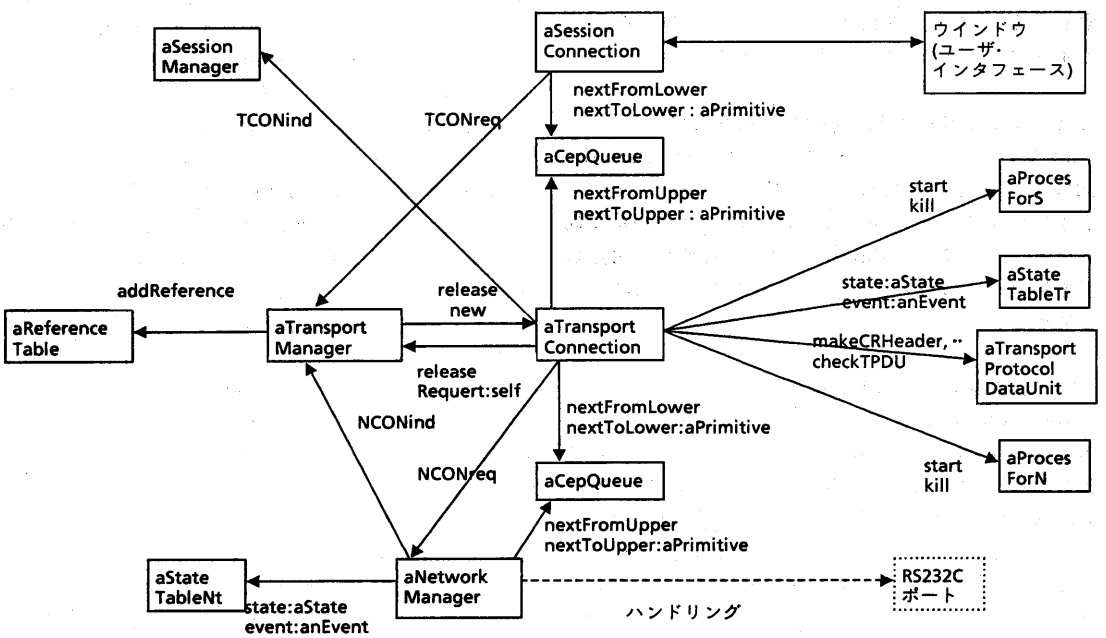

図2 オブジェクト間のメッセージの流れ

通な機能を上位のクラスとして定義し直して他のレーヤでも再利用できるように、全てのレーヤに共通なメソッドとトランスポートレーヤに特有なメソッドとは区別して作成した。図2に、今回作成したプログラムにおけるオブジェクト間のメッセージの流れを示す。

4.2 クラスTransportManager

管理用オブジェクトに対応するクラスであり、トランスポートコネクション(TC)の確立･解放を行う。コネクションのための確立･解放のメソッドは同時に実行されてはならないので、Semaphoreによる相互排除を行っている。また、TCの確立･解放に伴うレファレンスの管理も併せて行う。

インスタンス変数としては、レファレンスの管理用テーブル及び相互排除のためのセマフォを持つ。レファレンスの管理用テーブルとしてクラスReferenceTableを新しく定義した。

次に本クラスの動作を示す。

①コネクションの確立

セッションレーヤからの接続要求(TCONreq)及び、ネットワークレーヤからの接続指示(NCONind)のインタフェース･プリミティブを、メッセージとして受けた後、ソースレファレンスを決定し、TransportConectionのインスタンスを生成する。同時にプリミティブのパラメータ及びソースレファレンスをTransportConectionのインスタンスに渡す。

さらに、TCONreqの場合は、CepQueueのインスタンスとして、TCEPに対応するキューを生成し、そのポインタを戻り値として返す。NCONindの場合は、NCEPに対応するキューへのポインタがネットワークレーヤより引数として渡され、それも併せてTransportConectionのインスタンスに渡す。図3に、TCONreqを受信した時のメソッドを示す。

```
メソッドTCONreq:aParameter
  |aTCEPQueue aSourceReference|
  accessProtect critical:[
  aTCEPQueue ← CepQueue new.
  aSourceReference←referenceTable addReference.
  TransportConnection new
  FromSession:aTCEPQueue parameter:
  aParameter reference:aSourceReference. ]
  ↑aTCEPQueue
```

図3 メソッドTCONreq

②コネクションの解放

TCが解放される時に、TransportConnectionのインスタンスから解放要求をメッセージとして受け、TransportConnectionのインスタンスを消滅させ、対応するプロセスも終了させる。同時に、レファレンスをテーブルから削除する。

4.3 クラスTransportConnection

コネクション対応オブジェクトに対応するクラスであり、TCの有限状態機械としての動作を行う。実現する機能は、コネクションの確立･解放に伴う動作、プロセスを用いたコネクションの並列動作、及び状態遷移表に基づく動作の3種類である。プロセス、状態遷移表及びレファレンス、TPDUサイズ等のTCに関する情報をインスタンス変数として持っている。

①コネクションの確立･解放に伴う動作

管理用オブジェクトにより、このクラスのインスタンスが生成･消滅され、これがコネクションの確立･解放に対応する。

生成時には、そのコネクションのソースレファレンス、プリミティブのパラメータ、及び隣接するレーヤとの間のキューを受け取り、同時に以下に示すプロセスを開始する。

②プロセスを用いたコネクションの並列動作

TCでの並列動作は、ネットワークレーヤ及びセションレーヤのキューからのインタフェース･プリミティブの読み出しとそれに対する有限状態機械としての動作という一連の処理の繰り返しである。従って、これをそれぞれプロセスとして記述することによりTCでの並列動作を実現する。

これらのプロセスは、TransportConnectionが生成された時に生成され、同時に活性化される。コネクション解放時にはプロセスも消滅させる必要があり、インスタンス変数として保持される。

また、TransportConnectionでの有限状態機械としての動作は、非可分であるのでセマフォにより相互排除を行っている。このプロセスを開始させるためのメソッドを図4に示す。

```
メソッドstartProcessForN
  |event|
  networkProcess←
  [ [true] whileTrue:[
   [ event ←networkQueue nextFromLower.
  semaphore critical:[self receivePrimitive:event.].
       Processor yield.] newProcess.
  networkProcess resume.
メソッドstartProcessForS
  |event|
  sessionProcess←
  [ [true] whileTrue:[
  [ event ←sessionQueue nextFromUpper.
```

```
semaphore critical:[self receivePrimitive:event.].
    Processor yield.] newProcess.
sessionProcess resume.
```

図4プロセス開始用のメソッド

③有限状態機械としてのとしての動作

インスタンス変数として保持されている状態遷移表(クラスStateTableTrのインスタンス)に従って行われる。インタフェース·プリミティブ受信時に、状態名、インタフェース·プリミティブ名をインデクスとして状態遷移表を引くことにより、実行すべきメソッドを選択し、そのメソッドを実行することにより有限状態機械としての動作を行う。

4.4 クラスStateTableTr

トランスポートレーヤの状態遷移表を実現するクラスである。実行すべきメソッド名の表をクラスArrayedCollectionを利用して二次元配列として持ち、状態名、インタフェース·プリミティブ名をインデクスとして引かれる時にその実行すべきメソッド名を値として返す。状態名、インタフェース·プリミティブ名をこの二次元配列へのインデクスに変換するための表をクラスDictionaryにより実現している。

4.5 クラスCepQueue

CEPに対応するクラスであり、上位から下位へのキュー及び下位から上位へのキューをまとめて実現する。上下2方向のキューをインスタンス変数として持ち、それぞれはクラスSharedQueueのインスタンスである。上位または下位への書き込みのメソッド、及び上位または下位からの読み出しのメソッドを持つが、実際の書き込み、読み出しはSharedQueueに対して行われ、並列に動作するプロセスの同期はそのSharedQueueで取られている。

4.6 プロトコル·データ·ユニット(PDU)

今回の実装ではトランスポート及びネットワークレーヤが対象となり、トランスポートプロトコルデータユニット(TPDU)とネットワークプロトコルデータユニット(NPDU)の実現が必要であった。TPDUとNPDUでは、PDUとして共通な機能とそれぞれに固有な機能から成り、その共通な機能を上位のクラスとして記述することができる。但し、ネットワークレーヤではNPDUは直接RS232Cポート上に転送されるためクラスByteArrayにする必要があり、TPDUのみをPDUとして扱った。

4.6.1 クラスProtocolDataUnit

PDUを実現するクラスである。PDUにおいて基本的なバイト単位の操作、連結、分離等の機能を実現する。さらに、ユーザデータ及びNPDUをByteArrayとして扱うため、PDUとByteArrayとの間の変換を行う機能も必要である。

プロトコルにおいてPDUは、一般にバイト列で表現され、さらにヘッダの作成·削除、連結、分離等の操作が必要でありその長さは可変であることが望ましい。従って、バイト情報を持つByteArrayと、その中での情報の先頭と終わりを指すポインタを用いて実現した。作成したメソッドには以下のものがある。

①任意個のバイトへのアクセス、置き換え等のバイト単位の処理を行うメソッド。

②PDUの連結·分離のために、他のPDUを加えたり、その1部分をPDUとして取り出すためのメソッド。

③PDUにByteArrayを加えたり、PDUからByteArrayを取り出すためのメソッド。

図5に、例としてPDUの先頭の1部分を取り出すためのメソッドを示す。

```
メソッドgetPDU:aSize
  |aPDU|
  aPDU ←ProtocoDataUnit new:aSize.
  aPDU replaceFrom:1 to:aSize with:self
  aPDU size:aSize.
  ↑aPDU
メソッドreplaceFrom:i to:j with:anArray
  pduArray replaceFrom:(head+i) to:(head+j)
          with:anArray.
```

図5メソッドgetPDU

4.6.2 クラスTransportProtocolDataUnit

クラスProtocolDataUnitのサブクラスであり、TPDUを実現し、TPDUに特有なフォーマッティング及びフォーマットのチェックの機能を有する。

①フォーマッティング

CR,CC,DT,DR,ER TPDUのヘッダを作成するメソッドを持つ。ユーザデータを持つTPDUは、ヘッダ作成の後、ByteArrayとしてのユーザデータを書き込むことにより行われる。

②フォーマットのチェック

受信したTPDUのフォーマットのチェック及びパラメータを読み出すためのメソッドを持つ。読み出したパラメータは、インスタンス変数packetElements(クラスDictionaryのインスタンス)に設定される。

4.7 インタフェース･プリミィティブ

インタフェース･プリミィティブは、プリミィティブ名とパラメータ群から成るが、必ずしも全てのパラメータを持つ必要はない。従って、プリミィティブ名、パラメータ名がキーとなる辞書として、クラスDictionaryを用いて実現した。但し、接続要求と接続指示に関しては、管理用オブジェクトへのメッセージになり、パラメータのみをDictionaryとして持つ。

4.8 タイマ

TPクラス0には2個のタイマが必要であり、クラスFSMDelayを用いて実現した。FSMDelayではタイマ値とタイマ発火後の処理を記述したブロックを受け取り、指定した時間が経過した後、渡されたブロックをユーザプロセスと同じ優先度のプロセスとして開始させる。但し、発火後の処理は、そのコネクション対応オブジェクトに対応する、有限状態機械として動作をするプロセスを妨げてはならないので相互排除する必要がある。

4.9 クラスReferenceTable

レーヤ内の管理情報であるレファレンスのテーブルを実現するクラスである。レファレンスは重複が許されないので、クラスSetのサブクラスとして定義した。レファレンスを他のコネクションと重複しないように選択するメソッドも持つ。

4.10 ネットワークレーヤ

先にパソコン上に作成したTPクラス0プログラム[5]とRS232Cの回線上で通信させるため、ネットワークプリミティブを提供し、かつRS232Cポートを制御するネットワークレーヤが必要である。このネットワークレーヤの手順は、X.25レベル3を簡易にした手順である。またNPDUは直接RS232Cの回線上で送受されるので、受信したNPDUの区切りを示すためにベーシック手順に基づくブロック同期を用いた。ネットワークコネクションは1本のみサポートすることとした。

4.10.1 クラスNetworkManager

ネットワークレーヤを実現するクラスである。ここでは、サポートするネットワークコネクションは1本であるため、本クラスで管理用オブジェクトとコネクション対応オブジェクトの機能をまとめて実現している。またRS232Cポートの制御機能も実現する。

実行時には2種類のフェーズがあり、コネクションの確立前は管理用オブジェクトとしての動作を行い、確立後にはコネクション対応オブジェクトとしての動作を行う。従って、それぞれの動作に対応するメソッドを区別して作成することにより、TPクラス0プログラムの構成方法とほぼ同様な方法で作成できる。RS232Cポートからの受信も常に並列に行う必要があるので、RS232Cポートからの受信もプロセスにより記述する必要がある。

4.11 ユーザインタフェース

作成したTPクラス0プログラムを用いて実際に通信を行うには、ユーザへのインタフェースが必要である。このユーザインタフェースはセションレーヤに位置するために、管理用オブジェクトに対応するクラスSessionManagerとコネクション対応オブジェクトに対応するクラスSessionConnectionを作成し、ユーザとのインタフェースとしてクラスSessionConnectionのインスタンスをモデルとするビューであるSessionConnectionViewを作成した。

4.11.1 クラスSessionConnectionとクラスSessionConnectionView

ユーザとのインタフェースを実現するクラスであり、TCEPに対応するキューを介してTPクラス0プログラムとの間でインタフェース･プリミティブをやりとりできるようにする。SessionConnectionでは、TPクラス0プログラムからのインタフェース･プリミティブの読み出しはプロセスにより行い、TPクラス0プログラムへのインタフェース･プリミティブの送出はSessionConnectionViewでのポップアップメニューにより行うようにする。受信されたインタフェース･プリミティブはSessionConnectionViewのサブビューに出力される。

5.結果と考察

今回、一般性の高いOSIプログラムの構成法の確立を目的として、TPクラス0プログラムをXerox AIW1121上に実装した。筆者らが先にパソコン上に作成したトランスポートプログラム[5]と実際に通信を回線速度1200bps、TPDUサイズ128バイトで行った結果、プログラムが正常に動作することを確認した。

そのプログラム構成について考察した結果を、以下に示す。

5.1 作り易さ

TPクラス0に対するソースコードは約800行であり、筆者らが先に作成したC言語のプログラムと比較して非常に少なくてすんだ。それに伴い開発効率も高くなった。ソースコードが少なくてすんだ理由としては、①Smalltalk-80では、先に作成したプログラムが容易に再利用できる、②コネクションの確立･解放とオブジェクトの生成･消滅との親和性が良く、コネクションの管理に関するソースコードが少なかった等が考えられる。

また、Smalltalk-80のプログラムの機能をまとめて記述することによるモジュラリティの高さを利用することにより、途中での設計方針の変更に柔軟に対処することができた。

5.2 構成方法の他レーヤへの適用性

本稿で提案した構成方法により、TPクラス0プログラムを容易に作成することができた。また、ネットワークレーヤのプログラムも容易に作成することができ、本稿で提案した構成方法の有効性及び他のレーヤへの適用性の見通しを得た。

但し、本構成方法において、隣接するレーヤのコネクションは1対1に対応するため、TPクラス2, 3, 4で行われる多重化等のように、複数の上位のレーヤのコネクションが下位レーヤのコネクションに対応する場合を扱えない。従ってこの場合は本構成方法の拡張が必要である。

5.3 作成したクラスの再利用性

今回の実装では、クラスの継承の機能を積極的に利用しなかったが、各クラスに対して、上位のクラスとして記述し直すことにより、他のレーヤのプログラムにおいても再利用できるメソッドについて検討した。以下に、主なクラスについて検討した結果を述べる。

①TransportManager…インスタンス変数の初期化以外のメソッドを、他のレーヤのプログラムにおいても再利用できる。

②TransportConnection…コネクションの確立･解放に伴う動作及びプロセスを用いた並列動作を行うメソッドを、他のレーヤのプログラムにおいても再利用できる。

③CepQueue…このまま全てのレーヤにおいて利用できる。

④ProtocolDataUnit…全てのレーヤのPDUにおいて、親クラスとして再利用できる。但し、今回の実装ではTPDUを実現したのみであり、今後他のレーヤのPDUを扱うためには、それらの各PDUの間の変換が必要になり、その点を考慮する必要がある。

6 おわりに

本稿では、Smalltalk-80によりOSIの各レーヤを共通に構成する方法を提案し、TPクラス0に適用し、通信プログラムを容易に作成できることを確認した。最後に、日頃御指導頂くKDD研究所鍛治前所長、野坂副所長、小野次長、浦野情報処理研究室長に感謝します。

参考文献

[1]: Goldberg, A. and Ronson, D.,"Smalltalk-80: the language and its implementation",Addison-Wesley,1983.

[2]: 加藤,長谷川,鈴木,"オブジェクト指向言語Smalltalk-80によるOSIトランスポート･プロトコルの実装",61年度前期情処全国大会

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

[4]: 土居,瀬川,"Smalltalk-80による平行プログラミング",コンピュータソフトウェア,Vol.3,No.1,Jan.1986.

[5]: 加藤,鈴木,"パソコンへのOSIトランスポート及びセションプロトコルの移植",61年度前期情処全国大会