マルチメディア通信と分散処理 46－8
（1990．7．12）

代数的言語OBJによるプロトコルの形式記述
--- 層構造に則した汎用フレームワークに基づく方法 ---

石ケ森 正樹[+] 岡田 康治[++] 二木 厚吉[++]

[+]日本ユニシス株式会社
[++]電子技術総合研究所

OSI基本参照モデルの構造に則った、代数的仕様記述言語OBJによるプロトコルの形式的記述の一試案を提示する。

今回の記述では、OSIの1つの層を単位として記述を行なった。これは、プロトコル記述のフレームワークであり、多様なプロトコルがその中で動く場の記述といえる。ここでは時間の概念の取り扱いを行なっており、動作を時間的な推移の中で見ることができる。また、OBJの実行系を使用する場合には、動作の確認を行なうための一種のテスト手段ともなり、これによってプロトコルの記述の段階でいわばパフォーマンスの観点を含めた動作の確認を行なうことができる。

今回の記述では、このシミュレーションをよりリアルに行なうために、通信路における非決定性の概念の記述を行ない、非決定的に動作する通信路を持ったプロトコルの動作の確認を行なえるようにした。

Formal Description of Communication Protocols in an Algebraic Specification Language OBJ
— A Description Method with a General Framework based on Layered Architecture —

Masaki Ishigamori[+] Koji Okada[++] Kokichi Futatsugi[++]

[+]Nihon Unisys Co.,Ltd. 2-17-51 Akasaka,Minato,Tokyo 107,Japan
[++]Electrotechnical Laboratory 1-1-4 Umezono,Tsukuba,Ibaraki 305,Japan

We describe a method for the formal description in OBJ of communication protocols based on OSI's Layered architecture.

In this paper, we focus on the general structure of each layer of OSI. This structure leads us to the general framework for the description of the communication protocol. In this framework we can describe many kind of protocols. In addition, the concept of time-proceeding is explicitly described in this framework and we can see the protocol's behaviour in order of time. When we use the execution system of OBJ, this framework gives a means for testing the protocol description and make it possible to check the description from the point of performance.

Again in addition, we describe explicitly the non-determinism of the communication medium and make it possible to see protocols' behaviour to the non-deterministic error in the communication medium.

1. まえがき

分散システムにおいて高度な信頼性を実現するための手段として通信プロトコルの振舞いの定義を形式的に記述することが有効だとされ、種々の形式記述技法（Formal Description Technique:以下FDT）が開発されている。ISO（国際標準機関）のOSI（Open Systems Interconnection:開放型システム間接続）活動の一環として開発されたEste-lle[3]とLOTOS[4]はその例である。FDTを使用することにより、適合性テスト・検証の厳密化が可能となり、仕様から実現への工程の自動化の可能性も考えられる。

筆者達は代数的仕様記述言語OBJを用いた形式的記述法の開発を行なっている[6][7]。本レポートではOBJによるプロトコル記述の事例報告を行なう。

今回の記述のポイントは、次の点である。

(1) サービス提供者としての1つの層をプロトコル記述の単位と考え、これを記述するためのフレームワークを作成した。ここでは、プロトコルエンティティは、このフレームワークに与えられるいわばパラメータと考えられる。

(2) 上記のフレームワークの記述において時間の概念を陽に記述し、その取り扱いの仕組みを実現した。

(3) このフレームワークに対して与える通信路の記述において非決定性の概念を陽に記述した。

OBJの実行系を使用すれば、この記述で、パフォーマンスの観点を含めたプロトコルの動作の確認を行なえる。これによって、プロトコルの適切さの確認をある程度までシミュレートでき、設計時に有効だと思われる。

本レポートではまず、OBJ言語の紹介とプロトコルの記述に関する基本的な考え方を述べる（2節）。その後に、OBJ言語によるプロトコルの仕様記述における基本構造を説明し（3節）、この構造に則ったフレームワークの実際の記述を紹介する（4節）。この時に時間概念の取り扱いについても説明する。その後に、フレームワークに与えるプロトコルエンティティと通信路の記述の例を紹介し（5節）、これを与えたシステムによるプロトコルの動作の確認について述べる（6節）。

2. 仕様記述における基本的事項

2.1 OBJ言語の概要[1][2]

OBJは、順序ソート代数をモデルとした代数型言語であり、項書き換えを用いた実行系を有している。OBJの記述は、抽象データ型を代数でモデル化して定義することによって行なう。即ち、集合とその上の演算をパックして代数として定義する。これがOBJ記述の基本単位であり、オブジェクトと呼ばれ、'obj'というキーワードで宣言される。各オブジェクトでは複数のデータ型を定義することが許されており、また、定義されたデータ型同士の間に順序関係を定義することができる。データ型は'sort'で始まる構文により、集合間の順序関係は'subsort'で始まる構文によって定義される。

定義されたデータ型の上の演算は関数として表現され、'op'と'eq'というキーワードで始まる構文によって定義される。op構文では、関数の定義域と値域が宣言され、eq構文では、等式を用いて関数間の関係を定義する（変数を含んだ形で定義されるのが一般的である）。この等式は実行系で左辺から右辺への書き換え規則として解釈される。

OBJ記述はオブジェクトの定義の系列である。この系列で、あるオブジェクトは先行するオブジェクトを利用（参照）できる。この時、オブジェクトをパラメータ化して利用する機能がある。これは、仮想的なオブジェクトを作成し、この仮想オブジェクトを仮パラメータとするパラメータ化されたオブジェクトを作成し、それが使用している仮想オブジェクトに対応する実際のオブジェクトをパラメータ化されたオブジェクトに実パラメータとして与えることによって実例化（instantiate）し、新たなオブジェクトを生成する機能である。この仮想オブジェクトはセオリーと呼ばれ、実パラメータとなるのは通常のオブジェクトである。セオリーは、'theory'で始まる構文で宣言される。ここでは、実パラメータとなるオブジェクトが満たさなくてはならない条件が定義される。また、実パラメータの仮パラメータへの束縛の仕方を記述する'view'という記述単位もあり、ここで仮パラメータとして宣言されたものと実パラメータのオブジェクトで宣言されたものとの対応を定義できる。このようなオブジェクトのパラメータ化機能によって、オブジェクトの再利用を促進し、部品としての価値を高めることができる。

2.2 レイヤ構造に基づく通信システムのモデル

プロトコルエンティティの動作を見るためには、ある1つのサービスを提供するユニットを1単位としてとらえることが有用であると考え、ここから記述のための基本構造を案出した。基本構造を次の図1に示す。このユニット全体を以下では通信システムと呼ぶ。

通信システムは、2つのユーザと、2つのプロトコルエンティティ（PE）と、1つの通信路から構成される。ユーザとPE、PEと通信路との間に各々交信の点サービスアクセスポイント（SAP）があり、各々 n、n-1 層サービスプリミティブ（SP）をやりとりする。

n層サービスは、n層のPEとn-1層の通信路によって提供される。2つのPEと通信路によって構成され、

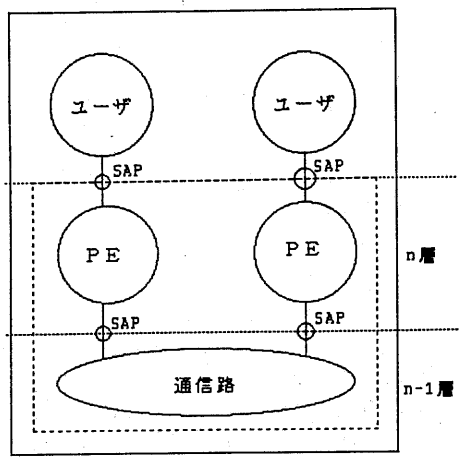

図1　通信システムの基本構造。

n層サービスを提供するユニットを本レポートでは層と呼ぶ。この観点からは、通信システムは2つのユーザと1つの層から構成されるシステムととらえられ、PEと通信路は層を構成する下部構造となる。

ところで、各層が提供するサービスの個別性は、その構成要素であるPEと通信路の個別性によって生じると考えられる。そう考えると、PEと通信路の個別性を捨象して考えた層自体は、PEと通信路がそこで動く共通な場としての一般的な性格を持つ。また、ユーザ、PE、通信路の個別性を捨象した通信システム自体についても同様である。これが今回の記述の基本的な観点である。

この観点からすれば、層自体に関する記述を具体的なPEと通信路に関する記述から独立させ、一般的な形で記述しておき、その一般的な層自体の記述に対してPEと通信路の記述を与えることによって具体的な層が得られる、という形で層を記述することができる。これは通信システムについても同様である。PE等の個別性を捨象した層自体と通信システム自体の記述は、具体的なPE等の記述がそこに与えられるフレームワークといって良い。

このようなフレームワークは、PEの動作の確認のための一種のシミュレーションの機構として使うことができる。この観点は、レイヤ構造の持つ再帰性を活かしたテスト機構を考える上で有効だろう。

2.3.　時間の扱い

通信プロトコルにおいては時間は重要なファクターだが、他のFDTの記述例においては、えてして時間に関する直接的な記述が行なわれておらず、タイマなどのプロセスの振舞いとして間接的に表現するか、注釈的に述べるかといったケースが多い[5]。しかし、時間の概念を直接に記述することは可能である。

今回は、タイムスタンプ型とでも言うべき方法で記述した[6][7]。SPのやりとりにはその時刻を示すタイムスタンプが押されると考える。これは、PEがSPの授受の瞬間毎に時刻を知ることができるという考え方に基づく。これは、SPの授受を表すデータ型に対して時刻を示すデータ型を加えたデータ型を定義することによって表現される。

今回の記述では、この考え方をフレームワークの記述のレベルで採用しておりSPのタイムスタンプを見て動作の割り当てを行なう機構も作成している。これについては4.2.2に詳細を記述してある。

このような機構の採用により、パフォーマンスも含めよりリアルな動作確認を行なうことができ、PEに与えるパラメータの類の適切値を実現以前に見積もることもできる。

3. OBJ記述の基本的構造

今回の仕様記述におけるOBJのオブジェクトの構成は、基本的には次の図2のような構成となる。

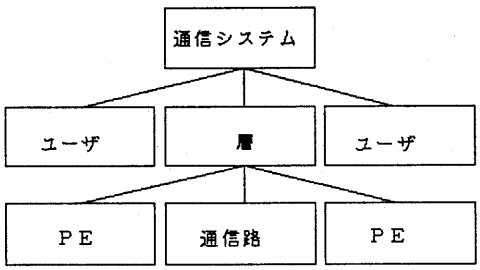

図2　オブジェクトの構成図（原則）

ユーザ、PE、通信路、層、通信システムの各々のオブジェクトで定義されるデータ型は、各々が各時点でそうなりうる様相もしくは状態の集合と考える（以下では「状態」という語に統一する。ここでの「状態」は、いわゆる状態遷移機械でいう状態ではなく、もっと一般的な意味である）。そこで定義されるトップレベルの関数は、ある状態が、そこから変化しうる最後の状態に時間的に推移することを表し、その意味で定義されたものの挙動を表す。

層の状態は、2つのPEと1つの通信路の状態の組み合わせであり、通信システムの状態は2つのユーザと1つの層の状態の組み合わせと考える。

この基本構造を実現するために、OBJのオブジェクトのパラメータ化の機能を使用した。今回の記述の構造は図3のようになる。図3は、5節で述べる実パラメータを与えた場合の構成を示している。

層の記述に則ってこれを説明するが、通信システムの記述でも基本的に同じである。

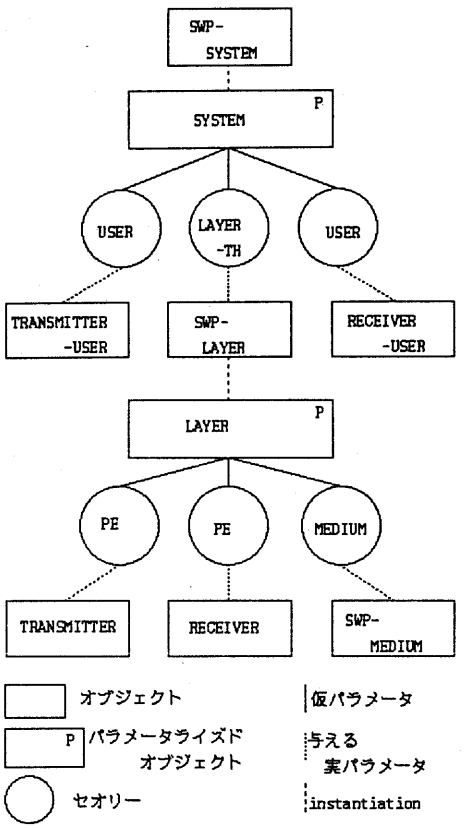

図3　オブジェクトの構成図（実際）

ここでは、層自体の記述がパラメータ化されたオブジェクトであり、これは、PEと通信路に関する一般的な（共通な）性質を記述した仮パラメータを使用して作成した一般的な層自体の性質に関する記述である。OBJの用語で言えば、セオリーとしてPEと通信路を表すモジュールを作成し、PEと通信路の一般的な性格を定義する。そして、これらのセオリーの記述を利用してパラメータ化されたオブジェクトとしての一般的な層の記述を行なっている。

各々の具体的なPEと通信路を表す実オブジェクトの記述は、仮パラメータとしてのPEと通信路の記述であるセオリーに対応した実パラメータとなる。これらの実パラメータの記述内容は、セオリーとして記述された仮パラメータで記述されている内容（条件）を満たさなくてはならない。

各々のオブジェクトの記述が作成されたならば、パラメータ化されたオブジェクトである層自体に対して実パラメータである具体的なPEと通信路の記述を与え、層自体の記述を実例化し、具体的で個別的な層の記述を生成する。

このような記述の基本構造をとることにより、層の記述に対する実パラメータとしてのPEと通信路の記述を差し替えることにより、様々なサービスを提供する層の形式記述が生成され、OBJの実行系を用いて動作を確認することができる。

4.　通信システムの記述

前節までに述べた考え方に基づいて作成した、通信システムに関するOBJによる実際の記述を提示する。この節では、フレームワークとしての層と通信システムの記述について説明する。これらの実例化については、次節で説明する。また、時間概念の取り扱いについてもこの節で述べる。

4.1　SPの記述

4.1.1　PEの入出力

PEは、ユーザと通信路双方とSPのやりとりを行なう。このSPのやりとりを各々OBJのデータ型として定義する。これらはPEにとっての入出力ととらえられ、計4種類の入出力になる。それに対応してオブジェクトも4つ定義する。今回の記述では各々をIN-FROM-USER、OUT-TO-USER、IN-FROM-MEDIUM、OUT-TO-MEDIUMという名称で定義している。ここでは、ユーザからの入力であるSPを表すオブジェクトIN-FROM-USERを例として示す。

```
obj  IN-FROM-USER  is  sort  InFromUser .
     pr    USER-DATA .
     op    ut : UserData -> InFromUser .
     op    ur : UserData -> InFromUser .
endo
```

演算子utとurは、各々ユーザとのSAPを表しており、2つのPEのどちらの入力かを区別する。ユーザから与えられるSP自身は、このオブジェクトの前に定義してあるUSER-DATAというオブジェクトでUserDataというデータ型として定義されており、このオブジェクトではそれを利用してPEへの入力を定義している。即ち、SPを表すデータ型をSAPを表す演算子でくくったものがユーザへの入力だと考えている。このオブジェクトで作成される項の例を次に示す。

例：ut('ConReq)

4.1.2　タイムスタンプ付きSP

前節で定義した入出力には各々タイムスタンプが付けられると考える。

時間の概念の表現自体はTIMEというオブジェクトで記述しており、オブジェクトTIMEでは、自然数として時刻を表現している。このTIMEオブジェクトを利用してタイムスタンプを表現する。

タイムスタンプの時刻は各々のゲートにSPが到着する時刻と考え、前節で定義したPEの入出力を基にして、それらと時刻を示す項とを演算子で括ったものとしてタイムスタンプ付きSPを定義する。

```
obj  INPUT-FROM-USER is sort InputFromUser .
     pr    IN-FROM-USER .
     pr    TIME .
     ...
     op    in-u  : InFromUser Time
                   -> InputFromUser .
     op    time : InputFromUser -> Time .
     ...
endo
```

例として、ユーザからPEへの入力を示す。それ以外に関しても事情は同じである。この例では、ConReqが入力となるSPを表し、13という数字がこのSPをユーザがPEに渡そうとした時刻を表す。

```
例 :in-u((ut('ConReq)),13)
```

4.2 層の記述

4.2.1 仮パラメータとしての通信路とPEの記述

層の一般的な性格を定義するパラメータ化されたオブジェクトLAYERにおいてその一般的な振舞いを定義するために、PEと通信路に要求される関数を仮パラメータ（セオリー）としてのPEと通信路の記述で定義する。即ち、ここで定義されるPEと通信路は、個々具体的なPEや通信路ではなく、PE一般に妥当する、もしくは通信路一般に妥当する性質を表現している。

一般的には次のようなものがある。

- PE
  PEの状態の集合を表現するデータ型
  ユーザとのSPのやりとり
  通信路とのSPのやりとり
  SPを受け取れる状態か否かをユーザもしくは通信路に知らせる
  ユーザもしくは通信路がSPを受け取れない場合に再送信する
- 通信路
  通信路の状態の集合を表現するデータ型
  PEとのSPのやりとり

尚、今回の記述では、通信路は各々違った方向の2つの通路から構成されるものとしてモデル化している。これで、少なくとも通信の基本的な動作は表現されうる。

```
th   PE  is  sort  Pe .
     pr    INPUT-FROM-USER .
     pr    OUTPUT-TO-USER .
     pr    INPUT-FROM-MEDIUM .
     pr    OUTPUT-TO-MEDIUM .
     op    iu : InputFromUser Time Pe
           -> Pe .
     op    ou : Pe -> OutputToUser .
     op    im : InputFromMedium Time Pe
           -> Pe .
     op    om : Pe -> OutputToMedium .
     ...
     op    available-iu : Pe -> Bool .
     op    available-im : Pe -> Bool .
     ...
endth

th  MEDIUM  is  sorts  Medium  Path .
    pr  INPUT-FROM-MEDIUM .
    pr  OUTPUT-TO-MEDIUM .
    op  m : Path Path -> Medium .
    op  put-pdu : OutputToMedium Time Path
                  -> Path .
    op  get-pdu : Path -> InputFromMedium .
    ...
endth
```

4.2.2 層の振舞いの記述と時間の取り扱い

層を表すデータ型Layerを定義し、このデータ型の構成子として、2つのPEと通信路と現在時刻を組み合わせる演算layerを定義する。これらの上で、層の振舞いを定義する。

層の振舞いとは、ある状態から他の状態へと時間的に層が変化することだととらえる。この変化は即ち、2つのPEと通信路の状態の組合せの系に生じる変化であり、2つのPEと通信路の各々の状態の変化に基づく。時間の経過とSPの到着によってPEと通信路の状態が変化し、それらの組み合わせである層が変化する。これが層の振舞いである。

層の振舞いを定義する関数は、一つの状態から時間的に変化しうる最後の状態を求める関数として定義される。これは、OBJの実行系においては、入力と層の（ある時点での）状態の組合せである項の（変化しうる最後の時点の）層の状態を表す項への書き換えとして実現される。

層の振舞いは次のような形式で示される。

```
transit(input, layer(pe1, pe2, medium, t))
        →layer(pe1', pe2', medium', t')
```

ここでpe1, pe2はPEを、mediumは通信路を表し、tはある状態の時刻を表す。transitは入力inputによって層の状態に変化を起こす演算子である。上記の形式は、入力inputによってPEと通信路のある時点の状態を表す項であるpe1, pe2, mediumの各々がpe1', pe2', medium' に変化し別の状態を表すようになることを表し、同時にそれらの組合せの層の状態も変化していることを表している（因みに、ここではpe等の出力は各々の構成要素としてそれ自身の中に含まれるものとして考えている）。tで表されている時刻もt'という変化が終了した時刻に変わる。

このような層の状態の変化を表現する関数に対し、時間的な制御を表現したり、SPのやりとりを表現するための補助関数が設定される。

時間的な制御は、PEと通信路の各々が出力するSPに付随する時刻を見て、ある時点で動作を行なうべきオブジェクトがどれかを判断して処理の割り振りを行なったり、現在時刻の設定を行なったりする幾つかの関数によって行なう。例えば、一方のPEであるPE-1が通信路に56というタイムスタンプのついたSP'data1'を出力しており、通信路がPE-2に48というタイムスタンプのついたSP'data2'を出力している時には、まず'data2'関する処理を行ない、次に'data1'の方の処理を行なう。この仕組みによって、独立なものとして動作するPEと通信路の動作を全体の中で配分することができる。

今回の記述においては、時刻は状態を表す各々の項に付随する項としてとらえているが、その時刻の変化は、PE等を表す項の変化（書き換え）に伴って時刻を表す項を書き換えるという形をとっており、これは、SPのやりとりに伴っている時刻を示す項の値を基にして行なわれる。今回の記述の対象の通信システムでは、複数のSPの入出力が並列的に生じる。この場合には、各入出力の瞬間の時刻を層のレベルで一元的に見て、優先順位を把握し、これに基づいて処理の割り振りを行なう。

以上が、PEと通信路の動きの場として捉えた層の記述の道具立てであり、これらによって、多種のプロトコルに妥当する一般的な枠組を記述する。

このような層の記述LAYERのOBJによる記述を以下に提示するが、紙面の都合で略記にとどめる。この記述では、入力を得る関数と、ある状態から最終の状態までの変化を生ずる関数とを分けている。これらを組み合わせることで上記の考え方が表現される。

```
obj  LAYER [ X Y :: PE , Z :: MEDIUM ] is
     sort Layer .
     ...
     op   layer : Pe.X Pe.Y Medium Time
                  -> Layer .
     op   l-transit  : Layer -> Layer .
     op   l-transit1 : Time Layer -> Layer .
     ...
     op   data-to-pe1 : InputFromUser Time
                        Layer -> Layer .
     op   data-to-pe2 : InputFromUser Time
                        Layer -> Layer .
     ...
endo
```

4.3 通信システムの記述

これらに関しては、枠組となる考え方は層の記述と同一であり、簡単に必要なことのみを述べる。

4.3.1 仮パラメータとしてのユーザと層の記述

ユーザと層を構成要素とする通信システムの全体系の定義にあたって、そのために必要なユーザと層の一般的な定義をセオリーとし定義する。

プロトコルの振舞いに関しては、ユーザはSPの蓄積場所（行き先と出所）という機能を有するのみであり、SPを表すユーザデータの系列としてとらえる。定義する関数は、層との間でSPのやりとりをする関数である。

```
th   USER  is  sort  User .
     pr    INPUT-FROM-USER .
     pr    OUTPUT-TO-USER .
     op    accept    : OutputToUser User
                       -> User .
     op    top-data  : User -> InputFromUser.
     ...
endth
```

層に関しては、ユーザとSPをやりとりするための関数を定義する。ここで定義される層は、あるユーザからSPを受け取り、それを他のユーザに渡す機能としての層である。この層という仮パラメータへの実パラメータは、前節のLAYERを実例化して作られるオブジェクトである。

```
th   LAYER-TH  is  sort  Layer .
   pr    INPUT-FROM-USER .
   pr    OUTPUT-TO-USER .
   pr    TIME .
   op request-u1 : InputFromUser Time Layer
                   -> Layer .
   op request-u2 : InputFromUser Time Layer
                   -> Layer .
   op out-of-pe1-to-u : Layer
```

```
                        -> OutputToUser .
  op out-of-pe2-to-u : Layer
                        -> OutputToUser .
  ...
endth
```

4.3.2 システムの記述

振舞いの表現に関する考え方、関数の定義に関する考え方は、層の記述と全く同一であり、定義される関数の種類も基本的に同一である。

ただし、ここでは、システムの中でやりとりされるSPは全てあらかじめユーザの中に設定されていると考えて記述を行なう。

従って、システムの遷移は次のような形式で表現される。

```
transit(sys (u1,u2,layer,t))
      →sys (u1',u2',layer',t')
```

ここで、u1,u2はユーザを表し、layerは層を表す。演算子sysは、ユーザと層を組み合わせてシステムの状態を表す項を作る構成子である。演算子transitは、システムの状態の時間的な推移を表す関数であり、システムのある状態からその変化しうる最終の状態までの振舞いを定義している。

このようなシステムにおける振舞いは、最終的にはユーザの状態の変化として見る。即ち、システムの初期状態ではユーザ1（もしくはユーザ2）にあったSPが、システムの最終状態ではユーザ2（もしくはユーザ1）に移動しているということである。

以上のような考えに基づくシステムのOBJ記述を以下に示す。紙面の都合でこれも略記とする。

尚、ここではシステムにおける状態の推移を観察しようという意図から、遷移の節目となる各ステップの状態を示す項をLogというsortの項として保持することにした。従って、最上位のsortは、システムの状態を示すsort System とLogとの直積であるSysLogというsortになっているが、これは便宜を計ってのことであり、基本的な構造の考え方とは関係ない。

```
obj  SYSTEM  [ A B :: USER , C :: LAYER-TH ]
     is  sorts  SysLog System Log . …
     op  slg : System Log -> SysLog .
     op  sys : User.A User.B Layer Time
               -> System .
     op  s-transit  : SysLog -> SysLog .
     op  s-transit1 : Time System Log
                      -> SysLog .
      …
endo
```

5. 具体的な層と通信システムの作成

ここまでで、パラメータ化されたモジュールとしての一般的な層とシステムの記述が終了した。この節では、前節での記述に具体的なPEと通信路の記述を実パラメータとして与えた、具体的な層と通信システムの記述の生成について述べる。

まず、実パラメータとして与えたものの説明を行ない、その際に、通信路の振舞いの非決定性の記述についても述べる。その後に、層と通信システムの記述の実例化について述べる。

5.1 具体的なPEと通信路の記述と非決定性の扱い

ここではSliding Window Protocol（以下SWP）の記述を示す[5]。SWPの基本構造を次の図4で示す。

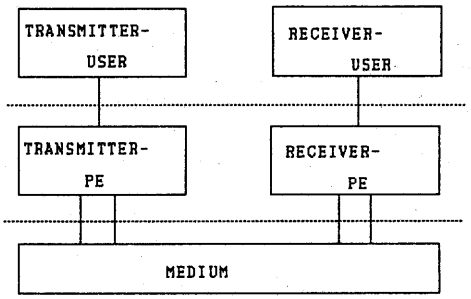

図4 SWP構造図

SWPでは2種のPEが設定されており、各々TransmitterとReceiverと呼ばれる。通信路ではPDUの転送異常が生じる可能性もある。

Transmitterを示すオブジェクトTRANSMITTERとReceiverを示すオブジェクトRECEIVER、並びに通信路を示すオブジェクトSWP-MEDIUMを作成した。システムに対する実パラメータとしては、ユーザを示すオブジェクトTRANSMITTER-USERとRECEIVER-USERを作成した。

SWP-MEDIUMの記述においては、通信路の非決定性を記述している。満足の行く回答とは言い難いが、ここでは乱数を利用して非決定性を記述した。乱数の値によって、通信路におけるPDUの転送が正常に行なわれたか否か、異常ならばどのような異常が生じたかを決定する。乱数の確率はOBJ記述において調整でき、これによって通信路でのデータ異常の発生の確立を制御し、何通りかの転送異常の発生率を設定してシミュレーションを行なうことができる。

以下に、乱数制御を行うモジュールを例示する。これ以外は、紙面の都合で省略する。

```
obj  COMMUNICATION-STATE is sort  ComState .
     pr    NAT .
     op    normal : -> ComState .
     op    discard : -> ComState .
     ........
     op   com-state        : -> ComState .
     op   error-state      : -> ComState .
     eq   com-state
          = if  (random-num < 60)
            then  normal
            else  error-state
            fi .
     .........
endo
```

5.2 層とシステムの記述の実例化

4節で定義した一般的な層とシステムの記述に対して、前節で定義したSWPの記述を実パラメータとして与えることによって、具体的なシステム（この場合はSWPのシステム）が生成される。これは、makeというコマンドで行なわれる。

今回の記述では、層LAYERとシステムSYSTEMの双方がパラメータ化されている。従って、instanti-ationも2重に行なわれる。即ち、まず層の記述をinstantiateし、そこでできた具体的な層の記述を使用してシステムのinstantiateを行なう。

最終的に生成されたSWP-SYSTEMというオブジェクトが、この例でのトップレベルのオブジェクトである。

```
make  SWP-LAYER  is   LAYER
       [ view to TRANSMITTER is
           sort  Pe  to  Transmitter ....
         endv,
         view to RECEIVER is ...endv,
         view  to  SWP-MEDIUM is ...  endv ]
endm

make  SWP-SYSTEM  is  SYSTEM
   [ view  to  TRANSMITTER-USER  is ...endv,
     view  to  RECEIVER-USER     is ...endv,
     view  to  SWP-LAYER         is ...endv]
endm
```

6. システムの実行とテスト

OBJの実行系を用いることによって、上記の記述の動作を確認することができる。

即ち、s-transit(sys)（sysはシステム）という関数の実行結果を見ることにより、システムがどのように振舞い、最終的にどのような状態にいたるかを知ることができ、記述した内容が意図通りのサービスを提供するものであるかを確認することができる。

また、各SPがユーザから層へ送られた時刻、層からユーザへ送られた時刻、セッションが終了した時刻も見ることができるので、記述したシステムのいわばパフォーマンスを知ることができる。これによって、PE等に与えるパラメータの類の適正値を求める目安を得ることも可能である。例えばSWPであれば、Transmitterの持つwindowのサイズやtimeoutまでのリミットの値など変えることにより、それらの適正値の目安を相対的にではあれ知ることができる。

次に実行の例を示す。

```
reduce in SWP-SYSTEM :
s-transit(slg(sys(t-user(Tinit ;
                  is(in-u((ut('data1)),1)) ;
                  is(in-u((ut('data2)),3)) ;
                  .........
                  is(in-u((ut('data9)),17));
                 is(in-u((ut('data10)),19))),
                  r-user(Rinit),
                  init-layer,
                  1),
               log))
result SysLog:
slg(sys(t-user(Tinit),
        r-user(Rinit ;
               os(out-u((ur('data1)),10));
               os(out-u((ur('data2)),12));
              ...............
               os(out-u((ur('data9)),188));
               os(out-u((ur('data10)),250))),
        layer(t-cnst((empty).
                  x-sys((empty),
                        NoPool,
                       timer(InitTimer,40),
                        x-win(10,11,2))),
              r-cnst((empty),(empty),
                  r-sys((NoOut),(empty),
                        nilArr,
                        r-win(10,11,2))),
              sm(path-tr,path-rt),
              258),
         258)),
    log;...1(
sys(t-user(Tinit ;
           is(in-u((ut('data10)),189))),
   r-user(Rinit ;
          os(out-u((ur('data1)),10)) ;
          os(out-u((ur('data2)).12)) ;
          .......
          os(out-u((ur('data7)).,122))),
```

```
layer(t-cnst((empty),
       x-sys((empty),
              put(9,'data9,put(8,'data8,
                  NoPool)),
              timer(put(9,180,put(8,180,
                  InitTimer)),40),
              x-win(9,8,2))),
      r-cnst((empty),(empty),
       r-sys((NoOut),(empty),
              nilArr,
              r-win(7,8,2))),
      sm(path-tr; p(unit('data8,8),188);
                  p(unit('data8,8),188);
                  p(unit('data9,9),188),
         path-rt),
      180),
 180))
 ...
```

ここでは、初期状態ではTransmitter側のユーザを表すt-userという演算子で括られていたSPの系列が、最終状態ではReceiver側のユーザを表すr-userという演算子で括られたSPとなっている。

layerという演算子で括られているのは、層を示す項であり、その中では、PEや通信路の状態が示されている。また、init-layerは、層の初期状態を示す項である。

また、resultの中のlogという所から後ろの出力は、最終状態になるまでどのような変化が起こったかを示す記録であり、中間の状態を表している。この例では、Transmitter側のユーザに'data10というSPがまだ残っており、Receiver側のユーザには'data1から'data7までが到着している。t-cnstという構成子で構成されているTransmitterのPEでは、バッファ内に'data8と'data9が蓄えられている。この2つのSPは、smという構成子で括られているTransmitterからReceiverへの通路に入っており、これは188という時点でReceiver側に渡るものだというタイムスタンプがついている。システムがこのような状態にある時刻は180という数字で示されている。

7. あとがき

本稿では、プロトコルの記述を行なうためのフレームワークをOSIのモデルに則した形で作成し、これが代数的記述言語OBJによってどのようにして実現されるかを、記述例と共に報告した。更に、時間の概念をフレームワーク中に組み込むことの意義と、それがOBJによってどのように実現されるかを報告した。また、OBJの実行系を用いて、このフレームワークの中でいかにしてプロトコルの記述の正当性を検定するかの一手法を、プロトコルの具体例とそれを使った実行例と共に示した。

今回採用したフレームワークは、OSIのモデルに基づいた通信プロトコルの共通の部分と個別的な部分とを分け、この分割によってより生産性の高いプロトコル開発技法を考えようとすることの1ステップである。このフレームワークの妥当性の検討が必要である。例えば、今回の記述ではPEの入出力を事前に定義して、これをフレームワークと具体的なプロトコルの双方で組み込みとしているが、入出力もフレームワークに対するパラメータと考えた方が自然なモデル化が可能かもしれない。

謝辞

本研究は、電子技術総合研究所における日本ユニシス（株）に対する技術指導の一環として行われたものである。本研究の機会を与えて下さった電総研棟上昭男情報アーキテクチャ部長、並びに日本ユニシス米口肇、深堀年弘両部長に感謝します。

References

[1] Goguen, J.A., Winkler, T.,: Introducing OBJ3, Technical Report SRI-CSL-88-9, SRI International,Computer Science Lab (1988).

[2] Futatsugi, K., Goguen, J.A., Meseguer, J., Okada, K.,: Parameterized Programming in OBJ2, Proc.of 9th Int'l. Conf. on S.E., (1987).

[3] ISO: Estelle: A formal description technique based on an extended state transition model, ISO 9074 (1989).

[4] ISO: LOTOS :Formal description technique based on the temporal ordering of observational behaviour,ISO 8807 (1989).

[5] ISO/IEC: Guidelines for the Application of Estelle, LOTOS, and SDL, ISO/IEC PDTR 10167 (1989).

[6] Okada, K.,Futatsugi, K.,: Supporting the Formal Description Process for Communication Protocols by an Algebraic Specification Language OBJ2 ,Proc.of 2nd Int'l. Symposium on Interoperable Information Systems, INTAP (1988) 127-134.

[7] 岡田康治, 二木厚吉 : 代数型言語による通信プロトコルの形式記述の一方法 , 日本ソフトウェア科学会第6回大会論文集 221-224 (1989).