Vol. 31 No. 10　　情　報　処　理　　Oct. 1990

解　説

# 形式仕様記述言語 LOTOS の試用経験†

大時和仁††　二木厚吉††

## 1. まえがき

異機種の計算機間通信を可能にするためには，一定の概念に基づいた標準的なネットワークアーキテクチャが必要であるとの認識に立ち，ISO（国際標準化機構）や CCITT（国際電信電話諮問委員会）を中心として OSI (Open Systems Interconnection)（開放型システム間相互接続）の基本参照モデルが制定され[1]，そのモデルに基づいて種々の通信規約の標準化が行われてきている[2)~4)]．

これらの規約は自然言語を用いて規定されており厳密に規定することが困難であった．そのために一つの規約に対して複数の解釈がなされたりして，相互通信ができないという問題が生じてきた．

そこで，通信システムのシステム・インタフェースの仕様を人工言語を使って形式的に規定し，解釈に曖昧さが残らないようにすることを目的として，形式記述技法 (Formal Description Techniques) の標準化が行われてきた．たとえば SDL が CCITT により，また Estell や LOTOS が ISO によりそれぞれ国際規格として制定された[5]．これらの形式仕様記述言語は，ISO で規定される現実の通信規約を記述対象として設計されている．

形式記述技法の導入には一般的には次のような利点がある[5]．

- システムの仕様を形式的な言語で記述することにより，仕様自体の無矛盾性，非曖昧性などの重要な性質を解析できる可能性が高い．
- 仕様の機械処理が可能であり，仕様の解析，蓄積，再利用などに計算機を有効利用できる可能性が高い．

特に上記の形式仕様記述言語に対しては，

- 通信規約に対して形式的な記述を与えることにより，その規約の中に含まれる矛盾のチェックをその形式的仕様記述に基づいて行うことや，
- 形式仕様記述に基づいてあらかじめテストケースを系統的に求めておき，実際にインプリメントされた通信プログラムをそのテストケースを用いて検定すること

などに対して精力的な研究がなされている[6]．

形式仕様記述言語はその基礎にある数学モデルに習熟しないと読みにくく，必要性は認識されていながらもなかなか現場までは普及しないのが現状である．しかしながら，上記の言語のうちで LOTOS は抽象データ型とプロセス代数という形式的な数学モデルに基づき実用を意識して設計された初めての言語でありながら，ISO の国際標準と制定されたという際だった特徴をもつ．また LOTOS は従来えてして役に立たない純粋基礎研究としてとらえられがちであった形式的な方法 (Formal Methods) を現場において評価する重要な機会を提供したともいえる．

われわれはこのような背景をもつ LOTOS の形式記述技法としての実用性と将来性を探ることを目的として幾つかの活動を行っている．本稿の目的の一つは，これらの活動において得られた LOTOS の試用経験を示すことにある．本稿では **3.** において，現場の通信プロトコルの技術者が LOTOS を実際に使ってみたときに抱いた感想を示すことにより，LOTOS の利点と欠点を評価する手掛りを与える．

さらに本稿では，**4.** において，LOTOS を OSI の通信規約記述のみならず，より一般的な問題記述に適用する可能性および LOTOS の改良点などについての展望を議論する．

**3.** で示す結果は「LOTOS 研究会」と名付けてわれわれが行ってきた研究会の活動によるものであり，次の 2 点について述べる．

- LOTOS を最初に勉強したときに抱いた感想，および
- 現実規模のプロトコルを記述してみたとき得られ

† Early Experience with a Formal Description Technique: LOTOS by Kazuhito OHMAKI and Kokichi FUTATSUGI (Electrotechnical Laboratory).

†† 電子技術総合研究所情報アーキテクチャ部言語システム研究室

た種々の問題点

一方，**4.**で示すものは，LOTOS 研究会とは独立に行っているものであり，通信規約の記述以外へのLOTOS の適用可能性，および，LOTOS の改良点について，LOTOS の将来展望について述べたものである．

ここで LOTOS 研究会について簡単に触れる．

われわれは，LOTOS のような抽象度の高い形式仕様記述言語が現実規模の通信規約の記述に耐え得るかどうかを調査するためには，形式仕様記述言語を研究所レベルで研究している人間と，実際に通信規約を製品として開発している人間とが一堂に会して検討を行うことが必要不可欠であると考えた．そこで，LOTOS の形式記述技法としての実用性を探ることを目的として，(財)情報処理相互運用技術協会(INTAP)に参加している中の有志各社の協力を得て LOTOS 研究会をつくり調査研究を行ってきた[17]．INTAP は通産省工業技術院の大型プロジェクト「電子計算機相互運用データベースシステムの研究開発」の委託研究を行っている法人である．

LOTOS 研究会は，INTAP の場を借りて，1989 年 1 月より 2 週に一回の割合で開催し 1990 年 3 月に終了した．参加会社は，(株)東芝，日本電気(株)，沖電気工業(株)，富士通(株)，三菱電機(株)，日本電信電話(株)，シャープ(株)の各社（順不同）であり，10 人前後の規模の研究会であった．メンバは，実際に OSI 通信規約の開発に携わっている人，プロセス記述の理論的な研究をしている人，形式記述言語に詳しい人，C言語などの言語プロセッサの開発に携わっている人，などであった．

LOTOS 研究会では通信規約記述言語に関して LOTOS 記述のガイドライン[11]の読み合わせ，簡単な例題（「哲学者の食事問題」）を書き，実際のプロトコル（ACSE[26]）の記述，LOTOS 処理系の検討などの活動を行ってきた．これらの活動の主要な結果に関しては参考文献 17)～21)にて報告した．本稿の **3.**ではこれらの成果のうち，LOTOS の記述実験に関することがらを述べる[17)~19)]．

## 2. LOTOS の概略説明およびその記述スタイルと OSI アーキテクチャとの関係

LOTOS 言語そのものの解説は文献 6)に詳しいのでそちらを参照されたい．この章では本稿に必要な最小限の説明を行う．

### 2.1 LOTOS の基本動作

LOTOS では，あるシステムの振る舞いを記述する際，そのシステムを外から見たときに観測できるイベントの生起順序を記述する．

イベントは「ゲート」と呼ばれる場所で生起できる．LOTOS では，イベントはゲート名の後ろにデータが付随したものである．これらのデータの型は，多ソート代数に基づく抽象データ型[9]（以下，ADT と略記する）を用いて定義する．LOTOS の ADT 部分は多ソート代数に基づく ADT 記述言語の一つである ACT ONE[8] に基づいている．ゲートにデータが付随したイベントの解釈については次節の例題を参照されたい．

イベントの生起順序を記述した表現を「動作表現」(behavior expression) と呼ぶ．

LOTOS における基本的な動作表現には次のように，CCS[10] を拡張した記法が使われている．ここに，$B, B_1, B_2$ は動作表現であり，$g_1, g_2, \cdots, g_n$ はゲート名である．

1. $a$; $B$ : action prefix

イベント $a$ が生起したあとの動作は $B$ によって規定される．

2. $B_1$[] $B_2$ : choice

$B_1$ か $B_2$ のどちらか一方の動作表現でイベントが生起できるとき，生起できたほうの表現が選択され，それ以降の動作がその選択されたほうの表現で規定される．

3. $B_1 |[g_1, g_2, \cdots, g_n]| B_2$ : parallel over $g_1, g_2, \cdots, g_n$

$B_1$ か $B_2$ のどちらかで生起できるイベントのうち，ゲート $g_1, g_2, \cdots, g_n$ で生起するイベントは同期して生起しなければならない．それ以外のイベントは interleave して生起する．

4. $B_1 ||| B_2$ : interleave

$B_1$ か $B_2$ のどちらかで生起できるイベントがあるとき，それらが interleave して生起する．

5. $B_1 || B_2$ : synchronize

$B_1$ か $B_2$ の両方で同時に生起できるイベントしか生起しない．

6. **hide**[$g_1, g_2, \cdots, g_n$]**in** $B$ : hiding

$B$ で観測されるイベントのうちゲート $g_1, g_2, \cdots, g_n$ で生起するものを外界から遮蔽する．

上記項目 4 番と 5 番は 3 番の特殊な場合である．同期演算子は CCS[10] における同期演算子 "|" と異なり，イベントが同期しても $\tau$ (外部から観測できない

```
specification TwoSlotBuffer [input, output]: noexit                    …(1)
  type NaturalNumber is                                                …(2)
    sorts Nat
    opns
      0:      →Nat
      s:  Nat →Nat
  endtype
  behavior                                                             …(3)
    hide middle in                                                     …(4)
      Buffer[input, middle]|[middle]|Buffer[middle, output]            …(5)
    where
      process Buffer[input, output]: noexit:=                          …(6)
        input?x: Nat; output!x; Buffer[input, output]                  …(7)
      endproc
endspec
```

**図-1** LOTOS による TwoSlotBuffer の仕様記述例

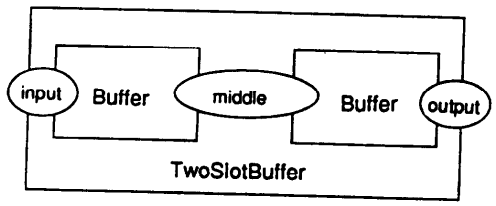


**図-2** TwoSlotBuffer の概念図

イベント）にはならず，外部から依然として観測可能である．6番の **hide** により外から観測不可能にする．

これら以外にも多数の演算子があるが，本稿ではこれ以上は言及しない．より正確な構文や意味については原典である文献 7)を参照されたい．

### 2.2 記 述 例

ここでは例を用いて LOTOS の基本的な機能を紹介する．

**図-1** が LOTOS を用いて TwoSlotBuffer と名付けたシステムを記述した例である．**図-2** がその概念図である．図-2 において，楕円形はゲートを表す．この例は文献 7)の Annex C より抜粋した．

図-1 において，行番号(1)はキーワード **specification** により，TwoSlotBuffer と名付けたシステムの仕様を記述することを宣言している．この行により，このシステムを外から観測するとき，input と output と名付けられた二つのゲートで生起するイベントを観測できることを記述している．input や output がゲートである(図-2 参照)．

キーワード **noexit** は，そのプロセスが終了することはないことを宣言している．

行番号(2)はこの仕様記述の中で使われる ADT の記述である．この例では NaturalNumber と名付けたデータ型を定義している．通常の多ソート代数に基づく ADT の場合と同様に，ソートやオペレータの定義を行っている．この例では用いていないが，キーワード **eqns** によってオペレータによって構成される項の間に成り立つ関係を等式で定義できる．

行番号(3)以下は TwoSlotBuffer は内部プロセスとして Buffer を用いる．

(4)，(5)行目を説明する前に Buffer の説明をする．

Buffer の定義は行番号(6)以下である．キーワード **process** により，プロセス定義であることを示している．Buffer* はゲート名として input および output をもつ．Buffer の動作表現が(7)行目である．

(7)行目では，input?$x$:Nat のイベントがゲート input で生起した後，イベント output!$x$ がゲート output で生起し，さらにその後の動作は，プロセス Buffer によって再び規定される，ということを記述している．すなわち Buffer は再帰的に定義されている．

二つのイベントのうち input?$x$:Nat は，ゲート input の場所で生起するイベントで，ソート Nat を値としてもつものすべてを表している．すなわち，input?$x$:Nat という記法は，input!0, input!$s$(0), input!$s$($s$(0)),… などのイベントを一括して表す省略記法である．

2番目の output!$x$ はこれらに応じて output!0, output!$s$(0), output!$s$($s$(0)),…などのゲート output において生起するイベントを表す．

Buffer の外側から観測されるイベントの系列を考えると，Buffer の外側の環境において input で起こるイベントと同期するもののみが選択されるので，イベントのこれらの可能性のうちどれか一つが選択されて外界から観測される．すなわち，直観的には Buffer は「$x$ に Nat 型の値が外界から与えられ，その値が output という場所で観測されるプロセスである」と考えてよい．したがって(7)の記述により，プロセス Buffer は「ゲート input に“入ってくる値”を $x$ で受けゲート output から“出す”」という動作を限りなく行う，と考えてよい．

上述した，直観的な意味でのイベント同期による値

* LOTOS では，何のシステムの仕様記述であるかを明示するために **specification** というキーワードを用いる．構造的にはキーワード **process** による定義とほぼ同等である．

の授受は，前節で示した同期演算子を用いて行われる．厳密には，？や！がそれぞれ入力・出力に使われるという言い方は正しくない．値の変域と値域の関係が整合していれば同期を起こす条件に当てはまり，！同士または？同士の組合せでも同期を起こす条件に合致し得る．この，同期の値渡しや，これらの他の演算子の詳細については文献 7)に譲る．

これらの規則により，図-1 の(5)行目は，二つの Buffer をゲート middle を介して直列に接続し，TwoSlotBuffer の input に入ってくる Nat の値を output に渡すことを表している．middle を介して Buffer を接続したことにより値が高々二つだけバッファされる．(4)の hide はゲート middle を外から観測できなくするための演算子である．

### 2.3 制約指向プログラミング

ここでは LOTOS における制約指向プログラミング (constraint-oriented programming) と呼ばれる記述スタイルについて述べる．ここで言う「制約」という用語は LOTOS の記述グループの間で用いられてきた記述スタイルに対する用語[12] である．「制約」という用語には他の用法もあるが[39]混同されないように注意されたい．

文献 11)に載っている Sliding Window Protocol の例を用いて制約指向プログラミングについて説明する．

このプロトコルの概念図を図-3 に示す．これは送信側 (Transmitter) と受信側 (Receiver) が通信媒体 (Medium) を介してデータを転送するプロトコルである．Transmitter と Receiver とはそれぞれゲート ut, ur を用いてこのプロトコルのユーザとデータのやり取りをする．すなわち ut! *d0* としてユーザは Transmitter にデータ *d0* を渡し，ur?$x$ : DataType として Receiver から $x$ に *d0* を受け取る．一方 Transmitter と Receiver とはゲート mt, mr により Medium を介してデータ転送を行う．

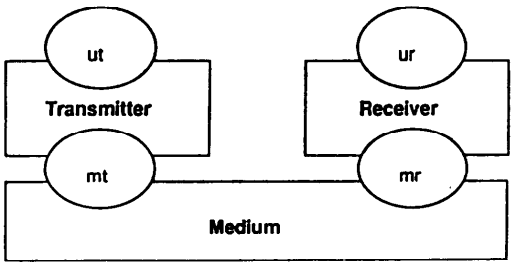


図-3 Sliding Window Protocol の概念図

このプロトコルの最初の部分を示したものが図-4 である．(1)行目は，このプロトコルがユーザに対してゲート ut, ur により交信することを示す．(3)行目と(4)行目は Transmitter と Receiver とがそれぞれ自分が生起できるイベントを interleave して任意の順序で生起できることを示す．

そして(5)行目の Medium とその直前の同期演算子 |[mt, mr]| とにより，(3)と(4)で interleave して起き得るイベント系列の集合のうちで，mt と mr でのイベントの順序に関して Medium による制約を満足するもののみが選択される．すなわち(3)，(4)で決まるイベント系列の集合に対して，mt と mr に関して(5)で制約を課していると考えることができる．

このような記述スタイルを「制約指向プログラミング」と呼んでいる．

文献 11)には記述スタイルとして推奨する技法が書かれている．そこでは上記のような制約指向プログラミングが，望ましいスタイルであるとして推奨されている．これは数多くの記述実験から得られた経験であり，巧妙でエレガントな書き方であるとされている．しかし巧妙過ぎて「具体的に理解した」という気になりにくいという批判もある．

### 2.4 OSI の 7 層モデルと LOTOS

LOTOS は OSI の階層構造の記述がしやすいように設計されている．OSI のプロトコルアーキテクチャは機能別に，ハードウェアに近いほうから，物理層，データリンク層，ネットワーク層，トランスポート層，

```
specification SlidingWindowProtocol [ut, ur]      …(1)
    behavior
        hide mt, mr in                            …(2)
            (
                Transmitter[ut, mt]               …(3)
            |||
                Receiver[ur, mr]                  …(4)
            )
            |[mt, mr]|
            Medium[mt, mr]                        …(5)
        where
            process Tramsmitter[ut, mt]
                …
            endproc
            process Reciever[ur, mr]
                …
            endproc
            process Medium[mt, mr]
                …
            endproc
endspec
```

図-4 Sliding Window Protocol の記述の一部

セション層，プレゼンテーション層，および応用層の7つの階層構造をなすものとして規定されている[1]．

OSIの各層におけるプロトコルの規定は，そのプロトコルがユーザに対してどのようなサービスを行うのかということを定義したサービス定義(service definition)および，そのサービスを下位のレイヤのもつサービスを用いてどのように実現しているかを記述したプロトコル記述(protocol specification)とからなる．

たとえば，図-3で示したSliding Window Protocolの例では，utとurがこのプロトコルの上位層に対するサービスを提供しているゲートである．またmtとmrが下位のレイヤから提供されるサービスを利用するためのゲートである．

この階層構造は前述の図-3と図-4で示したようにゲートを用いて階層構造が反映されるような記述がなされている．

### 2.5 関連研究

LOTOSに関しては，ヨーロッパのESPRITプログラムの中でSEDOS (Software Environment for the Design of Open distributed Systems)プロジェクトとして精力的に行われてきており，その成果が文献12)に載っている．LOTOSの理論的な成果および処理系などに関しては文献6)で解説されている．

ここではLOTOSを用いた記述実験の主な成果について述べる．

OSIアーキテクチャにおける通信プロトコルの仕様記述として，下位層のうちセション層が現在，テクニカルレポート[13),14)]としてISOの文書となっている．また，トランスポート層の記述もみられる[11)]．

現在は，下位層の記述よりもむしろ上位層(応用層)の記述に関心が集まっている．応用層のうちトランザクション処理のプロトコルで2nd DIS (Draft International Standard)の中にもLOTOSによる記述の試みが行われている[15)]．またCCRと呼ばれるプロトコルに対しても記述の試みがある[16)]．

これらの文書のうち，文献13)，14)はすでにISOにおけるTR(テクニカルレポート)になっており，(財)日本規格協会を通じて入手できる．また文献11)も90年10月にテクニカルレポートによる予定である．文献15)は現在DIS (Draft International Standard)の段階であるが，IS (International Standard)になれば，他の文書と同様に(財)日本規格協会を通じて入手できる．

正式なISOの文書とは多少異なるが，文献12)にもこれらの原型となる記述を見ることができる．

## 3. 記述実験

本章で記述する内容はLOTOS研究会のメンバの間で行ってきた検討内容である．

### 3.1 簡単な例題によるLOTOSの考察

LOTOSで実際の問題の記述を行ってみて感覚を掴むことを目的として，「哲学者の食事問題」を取り上げ，何人かで独立に記述してみて，LOTOSの記述経験を深めた[17)]．この記述はHIPPOと呼ばれるシミュレータ[22),23)]を用いて動作実験を行った．ここでは記述自体の説明よりもむしろ，この記述経験から得られたLOTOSの感想が大切であるので，それについて述べる．

ここで示すものは，特に，プロトコルを現場で開発している専門家が，LOTOSを初めて知ったときに抱いた感想を，率直にまとめたものである．用いた例題は非常に簡単ではあるが，その例題を元にして，「プロトコルを記述するとしたらどう思うか」という立場に立っての感想である．これらは今後，LOTOSを現実に使ってみようと考えている人々にとって有用な判断材料を与えるものと考える．

**[ADT部分の分かりにくさ]**

- 慣れていないせいもあるが，抽象データ型を書いたとき本当に思ったことが表されているかどうかに自信がもてない．また，読むときも，そのデータ型が何を表しているのかを，コメントなしには理解できない．
- プロトコル記述に向いた抽象データ型のライブラリの整備が必要である．特に応用層の記述においてはASN.1[27)]相当のデータ構造記述をADTでライブラリとして記述しておくことが是非とも必要である☆．

**[プロセス記述部分の分かりにくさ]**

- ？や！が入力にも出力にも使われるのは紛らわしい．これは，LOTOSが？と？同士，あるいは！と！同士でもイベントの同期条件を満たすことがあり得るように定められた言語であるためである．これはCSP[24)]に似た記法を採用しながら，LOTOSがCSPと大きく異なる部分である．
- 並列演算子が豊富にあるため間違えやすい．強力

☆ ASN.1は応用層のパケットのデータ構造を，拡張された正規表現を用いて定義するための表記法である．このライブラリに関しては3.3で説明する．

であるともいえるが，演算子の順位が少し複雑であり間違えやすい．

●特に再帰定義が含まれている同期プロセスの記述において，読んで理解するのに手間がかかる．

**［言語全般について］**

●LOTOS は読むのもそれほど楽ではない．読むためにも書くためにもなんらかの動くツールが是非とも必要である．

●しっかりした数学的基礎に基づいているので好きだ．

●並列に関する演算子が豊富であるため，表現したいことを素直に表せる．

●階層的記述（制約指向型記述）は記述する人にとっては楽に書ける．しかし読む人にとっては記述者の意図を読みとれない場合が多い．

●十分に LOTOS に関する記述経験を積んだ上でその(シミュレータなどの)処理系の設計にかかるべきであり，現状ではその経験は不十分である．LOTOS 導入によるメリットが今のところ必ずしもはっきりしない．

●一般に，ソフトウェアの仕様記述言語として LOTOS を利用する場合を考える．LOTOS を用いたとしても，いきなり詳細な仕様を LOTOS で書き始めることは困難である．開発工程において，抽象化的なレベルの LOTOS 記述から段階的に詳細化して徐徐に具体的なレベルの LOTOS 記述を得る，といった利用が妥当であると思える．

**［制約指向と状態指向について］**

制約指向型の記述だけでなく，状態遷移表がなんらかの意味で，LOTOS 記述に反映されるようなプログラミングスタイルも必要である．従来プロトコルに従事してきた人間にとっては，状態遷移表が馴染みやすいからである．LOTOS のプログラミングスタイルについては特に，制約指向型のスタイルと，状態指向型のスタイルとが対比されている[25]．

ここではまず，図-1 で示した例題を用いて，状態指向の記述スタイルについて説明する．

図-1 は，TwoSlotBuffer の仕様を，制約指向型で書いたとみなせる．なぜなら，二つの Buffer の生起できるイベントを，同期演算子 |[middle]| を用いてお互いに制約を課すことにより，生起可能なイベント系列の集合を規定している記述であるとみなせるからである．

一方，これと同等な記述を，状態遷移図を用いて記述した例が**図-5** である．この図において，$s0$ は初期状態であるとし，$x1$ や $x2$ などはソート Nat の変数であるとする．これらの変数は，TwoSlotBuffer が，Nat の値を二つだけ蓄えておくことができることを考慮して二つ用意した．この状態遷移図において，input?$xi$ のイベントが生起したとき，次に生起する output!$xi$ で $xi$ の値がゲート output において得られるものとする．

この状態遷移図に基づいて TwoSlotBuffer の記述を状態指向で行った LOTOS 記述が**図-6** である．この図の行番号(1)で“状態”に相当するデータ型を定義し，行番号(2)で状態として $s0$ から $s5$ までをソート State の定数項として定義している．

行番号(3)で以下で定義するプロセス FSM を初期化している．**2.** では説明を省略したが，LOTOS ではプロセスにパラメータを付すことができる．プロセス FSM はパラメータとして $s0$，$x1$，$x2$ をもっている．それぞれのソートは State, Nat, Nat である．行番号(3)では FSM が初期状態として $s0$ を入れ，$x1$，$x2$ の値として 0 を入れている．ただし，このうち $x1$ と $x2$ の初期値は 0 でなくても良いが，便宜上 0 に初期化してある．

そして，(4)の FSM が(3)から呼ばれたとき（正確にはプロセスインスタンシエイションと呼ぶ），$s$ の初期値が $s0$ であるのでまず(5)の動作表現が呼ばれる．[$s$ eq $s0$] などはガードと呼ばれ，[ ] の中の条件が満足されるとき，→に続く動作表現により次の動作が規定される．したがって，(5)は，input?$x1$:

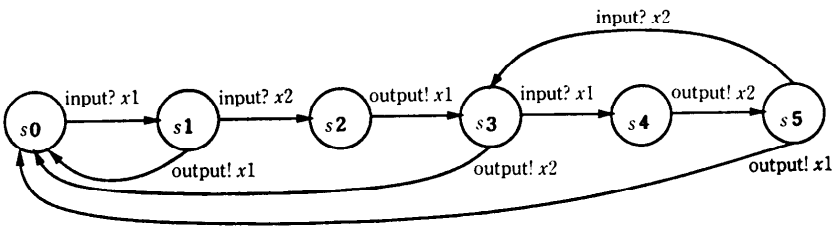


**図-5** TwoSlotBuffer を状態遷移図で記述した例

```
specification TwoSlotBuffer [input, output]: noexit
    type NaturalNumber is
    ...
    endtype
    type States is
        sort State                                                    ...(1)
        op s0, s1, s2, s3, s4, s5: →State                             ...(2)
    endtype
    behavior
        FSM[input, output](s0, 0, 0)                                  ...(3)
        where
            process FSM[input, output](s: State, x1: Nat, x2: Nat): noexit:=   ...(4)
                [s eq s0]→input?x1: Nat; FSM[input, output](s1, x1, x2)        ...(5)
            [] [s eq s1]→output!x1; FSM[input, output](s0, x1, x2)             ...(6)
                    [] input?x2: Nat; FSM[input, output](s2, x1, x2)           ...(7)
            [] [s eq s2]→output!x1; FSM[input, output](s3, x1, x2)
            [] [s eq s3]→output!x2; FSM[input, output](s0, x1, x2)
                    [] input?x1; FSM[input, output](s4, x1, x2)
            [] [s eq s4]→output!x2; FSM[input, output](s5, x1, x2)
            [] [s eq s5]→ouput!x1; FSM[input, output](s0, x1, x2)
                    [] input?x2; FSM[input, output](s2, x1, x2)
            endproc
endspec
```

図-6 TwoSlotBuffer を状態指向で記述した例

Nat のイベントを生起し，状態を $s1$ にして FSM を呼ぶ．このときパラメータ $x1$ は input?$x1$: Nat により入力された $x1$ の値となる．

同様にして，(6)と(7)ではそれぞれ FSM を状態 $s0$ および $s2$ に遷移させることを記述している．

以上が状態指向の記述スタイルの説明である．

多くのプロトコルの規約において状態遷移図が多用されており，したがってプロトコル技術者の多くが，制約指向よりも状態指向のほうが馴染みやすい．通信プロトコルの技術者は，与えられた問題のトップレベルの抽象化の段階では制約指向の記述スタイルをとり，詳細化されていったある段階からは，状態指向により記述することが望ましい，との感想をもっているようである．

### 3.2 ACSE の記述

LOTOS 研究会では OSI の応用層の規格である ACSE (Association Control Service Element) と呼ばれるプロトコル[26]を LOTOS により記述することを試みた[18]．この記述は，特に，内山光一氏（東芝）および藤田朋生氏（日電）の両氏によるところが大である．

#### 3.2.1 ACSE の簡単な説明

ACSE に関する説明はたとえば文献 2)，3)などが分かりやすい．ここではこれらを参考に，ACSE の簡単な説明を行う．

OSI の通信プロトコルを利用する応用プロセスは，通常，通信に関わる仕事だけではなく，開放型システム内のローカルな仕事も行っている．応用プロセスのうち，通信に関わる側面だけを取り出して，応用エンティティ (AE) と呼ぶ．

二つの AE の間で送受されるプロトコルデータは第6層のプレゼンテーション層を介して運ばれる．このときプレゼンテーション層によって運ばれる ACSE のプロトコルデータ単位 (PDU) は ASN. 1[27] によってその構文が定義されている．

AE 間で通信を行うための論理的な通信路をアソシエーションと呼ぶ．通常，下位層のプロトコルにおいてコネクションと呼んでいるものに相当する．応用エンティティ間のアソシエーションの確立および解放を司るのが ACSE である．

ACSE は以下の4つのサービス*を利用者に提供する．

A-ASSOCIATE 要求/指示/応答/確認
A-RELEASE 要求/指示/応答/確認
A-ABORT 要求/指示
A-P-ABORT 指示

A-ASSOCIATE はアソシエーションを確立するために用いられ，必ず確認がとられる．A-RELEASE はアソシエーションを解放するために用いられ，両方

* ACSE の利用者からみたときのシステム・インタフェース

の AE の合意に基づきアソシエーションは解放される．A-ABORT と A-P-ABORT はアソシエーションを強制的に解放するためのものであり，前者は ACSE サービスの利用者または提供者から起動される場合に使用され，後者はプレゼンテーション以下の層から起動された場合に使用される．

各サービス要素はそれぞれ以下のように PDU（プロトコルデータ単位）に対応付けされる．

A-ASSOCIATE 要求/指示⇒AARQ APDU
A-ASSOCIATE 応答/確認⇒AARE APDU
A-RELEASE 要求/指示 ⇒RLRQ APDU
A-RELEASE 応答/確認 ⇒RLRE APDU
A-ABORT 要求/指示 ⇒ABRT APDU
A-P-ABORT 指示 ⇒PDU は存在しない

典型的なプロトコルのシーケンスを図-7 に示す．

この図で示されているように，ACSE は応用プロセス間に論理的な通信路を張ったり，その通信路を種種の理由により解放したりするためのものであり，応用プロセスにおいて共通に用いられる．

### 3.2.2 ACSE を選んだ理由

記述対象として ACSE を選んだ理由は次のとおりである．

1. 下位層の記述はすでにいくつかなされている[13),14)]．応用層の記述の試みもあるが[15),16)]，応用層のプロトコルを完全に記述した例は公表されていない．

2. ACSE は応用層としては前節で示したとおり，仕様があまり複雑でなく，LOTOS による形式記述化がまだされていないので，記述実験に向いている．

3. 応用層は下位層と違って第6層のプレゼンテーション層との ASN. 1 記述が含まれるので，その部分の LOTOS による記述実験をすべきである．

ACSE プロトコルは，アソシエーションの確立と解放の処理だけを規定した OSI の中では比較的簡単なプロトコルであるが，

- プロトコルデータ単位のフォーマットを ASN. 1[27)] を用いて記述しているのでその部分が LOTOS の抽象型でどう記述されるかということ，および
- 応用層の中での ACSE の役割を明示できるかたちで記述すべきであるが，ACSE の記述というのは応用層のモデルの中でどの部分を記述することになるのかということの2点に特に興味をもって記述を行った．後者に関しては次節参照のこと．

### 3.2.3 記述した部分

一つの AE は複数の応用サービス要素 (ASE) といくつかの調整機能から構成される．ASE は特定の通信機能の集合であり，ACSE はアソシエーションを制御するための ASE である．

AE の内部構造は，ALS (Application Layer Structure)[28)] と呼ばれるモデルで規定されている．ACSE の論理的な位置づけも，この規定に従ってなされている．ここでは，ALS に従って ACSE の論理的な位置づけをもう少し詳しく説明する．

AE は複数のアソシエーションをサポートすることが可能である．その際，1本のアソシエーションに閉じた処理を行う部分を SAO (Single Association Object) として定義し，複数のアソシエーションに関する調整の機能は MACF (Multiple Association Control Object) が行うこととした．一つの AE の中には複数の SAO が存在することがあり，また一つの SAO は複数の ASE を内包している．たとえばファイル転送の機能をもつ ASE はコミットメント制御のための ASE やアソシエーション制御のための ASE (ACSE) とともに動作する．ASE 間の調整機能は SACF (Single Association Control Function) により制御される．

以上のことを図にして示すと図-8 のように示される．

ACSE の記述ということでは，SAO の中の ACSE の箱の部分だけを記述すれば良いのだが，たとえば FDT のガイドライン[11)]の中のトランスポート層の LOTOS 記述ではアドレスやコネクション端点の割当て／管理

| | | |
|---|---|---|
| A-ASSOCIATE 要求→ | —AARQ APDU→ | →A-ASSOCIATE 指示 |
| A-ASSOCIATE 確認← | ←AARE APDU— | ←A-ASSOCIATE 応答 |
| A-RELEASE 要求→ | —RERQ APDU→ | →A-RELEASE 指示 |
| A-RELEASE 確認← | ←RERE APDU— | ←A-RELEASE 応答 |

(a) 正常なアソシエーションの確立と正常なアソシエーションの解放

| | | |
|---|---|---|
| A-ASSOCIATE 要求→ | —AARQ APDU→ | →A-ASSOCIATE 指示 |
| A-ASSOCIATE 確認← | ←AARE APDU— | ←A-ASSOCIATE 応答 |
| A-ABORT 要求→ | —ABORT APDU→ | →A-ABORT 指示 |

(b) 正常なアソシエーションの確立と起動側 AE からのアソシエーションの強制解放

| | | |
|---|---|---|
| A-ASSOCIATE 要求→ | —AARQ APDU→ | →A-ASSOCIATE 指示 |
| A-ASSOCIATE 確認← | ←AARE APDU— | ←A-ASSOCIATE 応答 |
| A-P-ABORT 指示← | 障害 | →A-P-ABORT 指示 |

(c) 正常なアソシエーションの確立と下位の層の障害による強制解放

図-7 シーケンス例

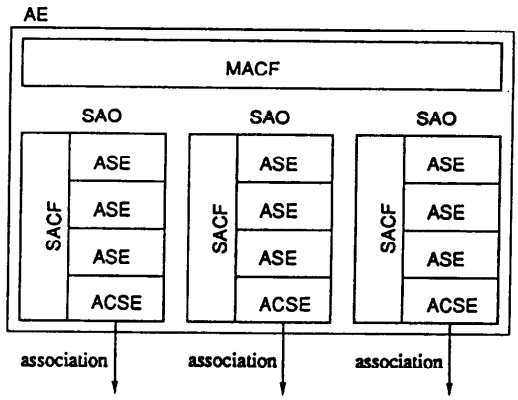


図-8 ALS に基づく AE の内部構造

といった，MACF や SACF のようななんらかの調整機能が行うであろう処理も記述されている．われわれはそれを参考にして MACF や SACF の記述も試みた．すなわち，図-8 の構造自身も LOTOS により記述した．

### 3.2.4 考 察

ALS 部分に関しては，ALS 規格[28]に明確に述べられていない部分や，アソシエーションの管理やプロセスの起動の仕方など実装に密接に関係する部分もあり，一般に記述が難しい．現在の記述は一つの解釈を与える感じになっている．そこで，このような解釈にあまり依存しない1本のアソシエーションを確立し解放する単純な正常系に関してのみ HIPPO[22]により動作を確認している．複数本のアソシエーションの確立や異常系に関しても動作は可能であるが，ALS のモデルに従っているかどうかに関しては，今後の検討が必要である．

プロトコルのシーケンスの記述については LOTOS 記述のほうが自然言語の記述や状態遷移表よりも理解しやすい．ただし，ACSE はプロトコルのシーケンスが比較的簡単であり，他のより複雑なプロトコルについても検討してみる必要がある．

記述を進める上で問題となったのは，どこまで詳細化すれば良いかという点である．規格の範囲を越えてしまわないように常に意識しておく必要があった．また ALS の記述の部分においては本来規格の中で規定されていなければならないものもあり，記述実験を行うことで，規格そのものを見直すいい機会となった．

## 3.3 ASN.1 のためのライブラリ

応用層の形式記述を LOTOS を用いて行う場合，考慮しなければならないのは，その応用層の規約の中の ASN. 1 で記述された部分である．応用層のデータ単位記述は規約中において ASN. 1 を使用して行われている．

ASN. 1 の解説が文献 41) にあり，言語仕様についてはそちらを参照されたい．

われわれ LOTOS 研究会では，応用層を LOTOS で記述するときには，この ASN. 1 部分を LOTOS の抽象データ型定義部分に対応させることが自然であると考えた．そこで，与えられた ASN. 1 記述に対して LOTOS の ADT 部分に変換する機械的手法を検討した．

ASN. 1 を ADT に変換する実験などに関しては文献30)，31)にみられる．われわれは，特に ACSE において実用的に使えるものを目指して詳細に検討を行った．

この変換方法の検討は，特に，五ノ井敏行氏（富士通）によるところが大である[19]．

### 3.3.1 変 換 例

ASN. 1 にはいくつかの型がある．われわれはそれら一つ一つに対して LOTOS の ADT への変換方法を考察しているが，ここでは紙面の都合上 ASN. 1の SEQUENCE 型の例を示す．

**図-9** は ACSE のリリースリクエスト (RLRQ) のプロトコルデータ単位 (PDU) の ASN. 1 記述部分である．この記述は直観的には，RLRQ という PDU が，リリースリクエストを出す理由と，ACSE の ユーザの情報とから成る長さ2の列 (SEQUENCE) の PDU であることを表している．これらの二つの値は，それぞれ reason および user-information をもちいて PDU から取り出すことができる．これらは，C や Pascal の構造体定義の中のフィールド名のような働きをする．

この ASN. 1記述を ADT に変換したものを**図-10**

```
RLRQ-apdu::=[APPLICATION 2] IMPLICIT SEQUENCE
  { reason                [0]  IMPLICIT Release-request-reason
                                  OPTIONAL,
    user-information      [30] IMPLICIT Association-information
                                  OPTIONAL
  }
Release-request-reason ::=…
Associtin-information ::=…
```

**図-9** RLRQ-apdu を ASN.1 により SEQUNCE を用いて記述した

```
type RLRQ_apdu is Boolean, Integer
  sorts RLRQ_apdu
  opns
        RLRQ_apdu:
          Release_request_reason__optional,
          Association_information__optional -> RLRQ_apdu
        reason:
          RLRQ_apdu -> Release_request_reason__optional
        user_information:
          RLRQ_apdu -> Association_information__optional
  eqns
    ...
endtype
```

図-10 RLRQ-apdu を ADT により記述した

に示す．この変換では，タイプ名やソート名が，もとの ASN. 1 記述から機械的に付けられている．また，PDU を合成する関数 RLRQ-apdu および PDU からそれぞれのフィールドの値を取り出す関数 reason および user-information が生成されていることに注意されたい．

### 3.3.2 考　察

ASN. 1 は抽象構文規則を定義する記法であり，形式的意味定義が与えられていない．したがって，記述の解析やその機械的処理のためのベースが与えられていないという欠点がある．一方，LOTOS は形式的な意味が与えられている言語である．特に，LOTOS の ADT 部分には，商項代数 (quotient term algebra)[9] により，形式的意味が与えられている．このことは，ここで用いた手法で変換を施すことにより，ASN. 1 に形式的な意味を与えることが可能になることを意味している．すなわち，ASN. 1 を LOTOS の ADT に変換し，この ADT に与えられる形式的意味を ASN. 1 の形式的意味と考えるのである．

形式的意味が与えられていない表記法である ASN. 1 に対して，形式的な意味を与えることにより，ASN. 1 を用いた応用層の仕様の意味が厳密になる．われわれが検討した手法はそのための試みである．

## 4. LOTOS の応用と将来展望

抽象データ型やプロセスは通信プロトコルに固有の概念ではなく，プロトコル以外の一般的な問題においても内包されている．したがって，LOTOS は通信プロトコル記述のために制定された言語ではあるが，より一般的な問題の仕様記述に対して，LOTOS で採用されている記述方法は有用であると考えられる．

ここでは通信プロトコルから離れて，LOTOS で用意されているような，プロセスとデータ型を枠組として与えられたとき，ソフトウェアの入出力関係に基づくモジュール間依存関係[37]とかプロセスプログラミング[36]などのソフトウェア工学の観点からどのような発展の可能性が考えられるかを述べる．

さらに，LOTOS をより高度に発展させるための可能性について，特に ADT 部分の拡張に関する考察を加える．

### 4.1 入出力関係に基づくソフトウェアの階層構造記述

ソフトウェアのモジュールの構造を入出力関係に基づき階層的に表すことにより，その振舞いが理解しやすくなることが多い[37]．

$M_i$ をソフトウェアのモジュールとし，in, out はそれらのモジュールの入出力（インタフェース）を表すものとする．多くの場合，ソフトウェアシステムの記述として，モジュールとその直下のサブモジュールの間の階層関係を，入出力の間の関係だけで表すことで有効な階層構造が得られる[37]．

モジュール間に成り立つ関係を表そうとするとき，単に入出力の間の関係だけでなく，モジュール間の時間的関係（時間的に独立であるとか，同期している必

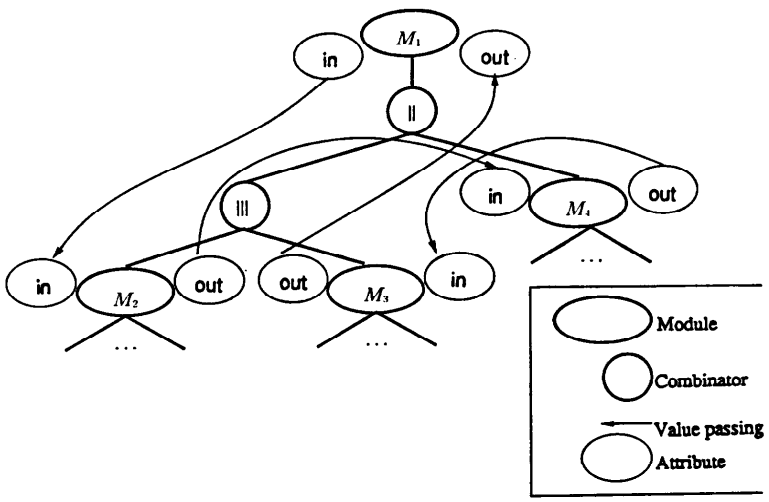


図-11 階層の間の値の授受

要があるとかいった関係）を表す機能があれば，モジュール間の関係に対してより豊富な記述の枠組を与えることができる．

**図-11** は，モジュール $M_1$ とその直下のモジュール $M_2, M_3, M_4$ の間になり立つ，入出力関係，および時間的関係を階層的な図で表したものである．

ここで，LOTOS のプログラムとして表されていたゲートは，階層間のデータの授受に使われ，また‖やなど|||の並列演算子は，モジュールの結合方法に対応しているものとする．

LOTOS で用いられている表記方法を用いると，モジュール間の時間的関係の有効な記述ができる可能性がある．

### 4.2 プロセスプログラムへの応用

ソフトウェア開発過程の記述と再利用は，ソフトウェア工学における最も挑戦的なテーマである[36]．

開発過程の記述はなされていないが，ソフトウェア開発の上流行程のレベルからコードレベルまでの開発に際し，LOTOS を意志疎通の道具として用いて設計し，最後のコードのところだけ Modula-2 に落とす，というプロジェクトを行ったという報告がある[29]．これは LOTOS で提起された枠組がソフトウェア開発一般にも広く適用可能であることを示唆している．

**図-12** は edit, compile, execute というプログラム開発の過程をプログラマの名前とともにイベントとして記述してみたものである．このように，LOTOS の並行プログラミング機構を用いて開発プロセスを記述することも可能であろう．

### 4.3 LOTOS の ADT 部分の改良に向けて

LOTOS で採用している ADT 部分は，必ずしも最新の ADT の成果を利用してはいない．そのために，使いにくいところが多い．この節では，代表的な代数的仕様記述言語である OBJ[32)~34)] との比較に基づく LOTOS の ATD 部分の改良点の候補について述べる．

OBJ は代数的仕様記述技法に基づいて主に ADT の記述を想定して設計された言語であり，その大きな特徴として次の 2 点をあげることができる．

1. パラメータ化プログラミングをサポートしている．
2. 順序ソート代数[35]に基づいてソート間の包含関係が定義できる．

この節では，順序ソート代数の概念を

```
specification Univ_PE[output](name : Name) : noexit
 type Name is
  sorts Name
  opns Kazu, Nori, Shio, Ayu, Juli :→Name
 endtype
 type Actions is
  sorts Actions
  opns edit, compile, execute : →Actions
 endtype
 behavior
  First_Action [output](name)
 where
 process First_Action[output](pn : Name) : noexit :=
  output!pn!edit ;
   (Second_Action[output](pn) []
   First_Action[output](pn))
  where
  process Second_Action[output](pn : Name) : noexit :=
   output!pn!compile ;
    (Second_Action[output](pn)[]
    Third_Action[output](pn) []
    First_Action[output](pn))
   where
   process Third_Action[output](pn : Name) : noexit :=
    output!pn!execute ;
     (Third_Action[output](pn)[]
     Second_Action[output](pn)[]
     First_Action[output](pn))
   endproc (* of Third_Action *)
  endproc (* of Second_Action *)
 endproc (* of First_Action *)
endspec
```

**図-12** プログラムの edit, compile, execute のサイクル

LOTOS の ADT 部分に導入することにより，記述が簡明に行えることを示す．この概念は ADT におけるエラー処理や例外処理を簡明に表すためにも有効である．

#### 4.3.1 順序ソートについて

あるソートで可能な操作が，別のソートでも可能であるような場合が多く考えられる．順序ソートはそのような場合を扱うこともできる．たとえば，自然数で

```
obj STACK is
 sort Any. — Elements of a stack
 sort NeStack Stack. —Non-empty stack and stack
 subsort NeStack<Stack. —Non-empty stack is a subsort of stack.
 op empty : →Stack.
 op push : Any Stack→NeStack.
 op top : NeStack→Any.
 op pop : NeStack→Stack.
 var A : Any.
 var S : Stack.
 eq top(push(A, S))=A.
 eq pop(push(A, S))=S.
endo
```

**図-13** subsort を用いた STACK

許される操作は整数でも許され，また整数で許される操作は有理数でも許される．このようなことを表すのに OBJ では次のように宣言する．

**subsorts** NatNum<Int<RatNum.

ここで，NatNum, Int, RatNum はそれぞれ自然数，整数，有理数のソートを表すものとする．

**図-13** はサブソートを用いて STACK を定義した例である．この例では空要素 empty を含むソートが Stack となるように，また含まないソートを NeStack となるように定義している．サブソートの関係は NeStack<Stack である．

すなわち push 操作を施すと NeStack のソートになり，top と pop は empty を含まないスタック NeStack に対してのみ定義されており，それに対して操作が行われる．

もしサブソート関係を定義できないとすると，empty のスタックに対する top と pop の操作に対して例外処理（エラー処理）に相当する等式などを用意しなければならず，記述が複雑になる．

### 4.3.2 LOTOS で順序ソートが使えたら

LOTOS では順序ソートの概念が用意されていない．この節では，もし LOTOS で順序ソートが使えたとしたら簡明に記述できるという例を示す．

**図-14** の(a)はオリジナルの LOTOS のプログラム例である．これは，ソート Integer の値 $x$ の正負に応じて，判定文 [$x$>0] と [$x$<0] によってゲート sap（サービスアクセスポイント）に出力する値を切り替える，という記述である．

もし，この Integer の記述に対して，図-14(b)のように非負整数 NNInt と負整数 NInt と Int との間にサブソートの関係が記述できたと仮定する．こうすると(b)に示すように正負の判定は ADT のソート名によって行われることになり，(a)と比べて簡明に記述できる．

一方，**図-15** は，前述の Stack の例を用いたものである．これは，優先データ転送のプロトコルの記述で用いられるものの一部である．図-15 の(a)は順序ソートを用いない場合で，empty に対して pop 操作を施そうとしているかどうかを [$x$ ne empty] でチェックしている．(b)では順序ソートを使い，この判定は ADT のソート判定の機構に任せてあり，記述が簡明になっている．

```
specification EX1 : noexit
 type Integer is Boolean
  sorts Int
  opns _<_ : Int, Int→Bool
       _>_ : Int, Int→Bool
       -_ : Int→Int
  ...
 endtype
 behavior
  inp?x :Int;
    (
     [x>0]→sap!x ; P[...](x, ...)
    []
     [x<0]→sap!-x ; P[...](X, ...)
    )
endspec
```

(a) オリジナル LOTOS

```
specification NEW_EX1 : noexit
 type Integer is Boolean
  subsort NNInt<Int
  subsort NInt<Int
  ...
 endtype
 behavior
  inp?X : NNInt ; sap!x ; P[...](x, ...)
 []
  inp?x :NInt ; sap!-x ; P[...](x, ...)
endspec
```

(b) もし LOTOS で subsort が使えたら...

**図-14** Integer で subsort を使う例

```
specification EX2 : noexit
 type Stack is
  sort Stack
(* 従来の Stack *)
  ...
 endtype
 behavior
  inp?x :Stack ;
   (
    [x ne empty]→sap!pop(x) ;...
   []
    sap!push(e, x) ;...
   )
endspec
```

(a) オリジナル LOTOS

```
specification NEW_EX2 : noexit
 type Stack is
  subsort NeStack<Stack
(* サブソート付の Stack *)
  ...
 endtype
 behavior
  inp?x :NeStack ; sap!pop(x) ;...
 []
  inp?x :Stack ; sap!push(e, x) ;...
endspec
```

(b) もし LOTOS で subsort が使えたら...

**図-15** Stack で subsort を使う例

## 5. あ と が き

LOTOS のプログラミングスタイルについては文献11)に 4 つの例題を用いて示されているが，必ずしも確立しているようにはみえない．文献 11)のスタイルは特に ADT 部分の定義は必ずしも最良のものとは考えられない．今後十分に現実のプロトコルを LOTOS で記述してみてその記述方法を叩いてプログラミングスタイルについて検討を積み重ねなければならない．

LOTOS の言語仕様も検討の必要がある．ASCE[26]の記述を行ってみてたとえばプロセスのスコープの概念とかゲートの動的割り付けなどの機能も必要であると感じた．これは将来の LOTOS の改訂作業において重要な問題提起になると思われる．また．応用層より上の，より抽象的な概念である ODP (Open Distribute Processing)[40] の記述においても LOTOS の利用が試みられており，その検討の中から，LOTOS にオブジェクト指向の概念を導入してコンパクトに記述できるようにしたい，という提案もなされている．現在の LOTOS の仕様では，簡単なプロトコルでさえもその LOTOS による記述が非常に大きくなってしまう．

LOTOS の ISO 標準の開発過程から類推すると，特に ADT 部分に関しては検討が必ずしも十分には行われてこなかったようにみえる．われわれは OBJ で用意されている種々の枠組を積極的にとり入れるべく検討を行っている[38]．

**謝辞** 査読者からは本稿を読みやすくするための改良点に関するご指摘を多数いただいた．ここに記して感謝する．

電子技術総合研究所情報アーキテクチャ部棟上昭男部長には LOTOS 研究会に対して支援をいただいた．本稿の **3.** は LOTOS 研究会のメンバによる検討結果である．LOTOS 研究会は多くの方々のご協力により開催できた．各社の関係者の方々には，LOTOS 研究会に対して快くメンバを出していただいた．(財)情報処理相互運用技術協会には研究会開催について便宜を図っていただいた．これらの方々に対してここに記して感謝する．

## 参 考 文 献

1) ISO 7478 : Information Processing Systems—Open System Interconnection—Basic Reference Model, 1984-10-15.
2) 田畑他：開放型システム間接続 OSI—明日へのコンピュータネットワーク，(財)日本規格協会 (1987).
3) 棟上他：開放型システム間接続 OSI の応用—高度情報化の実現に向けて，(財)日本規格協会 (1987).
4) 大特集：ネットワークアーキテクチャ(開放型システム間相互接続) の標準化動向，情報処理，Vol. 26, No. 4 (1985).
5) 二木：ISO における形式記述技法の標準化動向 情報処理，Vol. 31, No. 1, pp. 3-10 (1990).
6) 高橋，神長，白鳥：LOTOS 言語の特質と処理系の現状と動向，情報処理，Vol. 31, No. 1, pp. 35-46 (1990).
7) ISO 8807 : Information Processing Systems—Open System Interconnection—LOTOS—A Formal Description Technique Based on the Temporal Ordering of Observational Behavior, 1989-02-15.
8) Ehrich, H. and Mahr, B. : Fundamentals of Algebraic Specification 1, Springer Verlag, Berlin (1985).
9) 稲垣，坂部：抽象データタイプの代数的仕様記述法の基礎(1)-(4)，情報処理，Vol. 25, No. 1, 5, 7, 9 (1984).
10) Milner, R. : A Calculus of Communicating Systems, Lecture Notes in Computer Science, 92, Springer-Verlag (1980).
11) ISO/IEC JTC 1 N 642 : Information Processing Systems—Open System Interconnection—Proposed Draft Technical Report on Guidelines for the Applicaton of Estelle, LOTOS, and SDL, 1990-01-18.
12) van Eijk, P. H. J., Vissers, C. A. and Diaz, M. eds. : The Formal Description Technique LOTOS, North-Holland (1989).
13) ISO/IEC/TR 9571 : Information Technology—Open Systems Interconnection—LOTOS Description of the Session Service, 1989-9-15.
14) ISO/IEC/TR 9572 : Information Technology—Open Systems Interconnection—LOTOS Description of the Session Protocol, 1989-9-15.
15) 2nd DP, ISO/IEC DP 10026-1, 2, 3 : Information Processing Systems—Open Systems Interconnection—Distributed Transaction Processing, Part 1 (Model), Part 2 (Service definition), Part 3 (Protocol specification), ISO/IEC JTC 1/SC 21/WG 5 (1988).
16) Ajubi, I. and van Sinderen, M. : High Level Synchronization Services of OSI—Comittment, Concurency and Recovery, the University of Twente MEMORANDOM INF-88-35 (1988).
17) 大蒔，五反田，小野，佐藤，藤田，新田，田中，堀田，五ノ井，内山，島田：哲学者の食事問題の LOTOS による記述実験，情報処理学会ソフトウェア工学研究会，66-4, 1989 年 6 月 27 日.

18) 内山, 藤田, 小野, 佐藤, 五ノ井, 田中, 辻, 山中, 大蒔：ACSE の LOTOS による記述の試み 情報処理学会マルチメディア通信と分散処理研究会資料 46-1, 1990年7月12日.
19) 五ノ井, 田中, 内山, 佐藤, 小野, 藤田, 山中, 辻, 大蒔：ASN. 1 から LOTOS ADT への変換法, 情報処理学会マルチメディア通信と分散処理研究会資料 46-2, 1990年7月12日.
20) 辻, 佐藤, 内山, 小野, 五ノ井, 田中, 藤田, 山中, 大蒔：クライアント／サーバ・モデルに基づく LOTOS 仕様記述支援システムの設計, 情報処理学会マルチメディア通信と分散処理研究会資料 46-3, 1990年7月12日.
21) 佐藤, 辻, 内山, 小野, 五ノ井, 田中, 藤田, 山中, 大蒔：LOTOS の汎用的な中間言語, 情報処理学会マルチメディア通信と分散処理研究会資料 46-4, 1990年7月12日.
22) Software Environment for Design of Open Distributed Systems, HIPPO-LOTOS Simulator, University of Twente, The Netherlands.
23) van Eijk, P. H. J.: Software Tools for the Specification Language LOTOS, Doctoral Dissertation, the University of Twente (1988).
24) Hoare, C. A. R.: Communicating Sequential Processes, Prentice-Hall Inc. (1985).
25) ISO/IEC JTC 1/SC 21/WG 1/N 843: Revised Working Draft for Architectural Semantics for FDTs (Sep. 1989).
26) ISO 8649, 8650: Information Processing Systems—Open Systems Interconnection—Service Definition (Protocol Specification) for the Association Control Service Element (1988).
27) ISO 8824: Information Processing Systems—Open System Interconnection—Specification of Abstract Syntax Notation One (ASN. 1) (1987).
28) ISO/IEC 9545: Information Technology—Open Systems Interconnection—Application Layer Structure, 1989-12-15.
29) Proc. of International Symposium on Protocol Specification, Testing, and Verification, North-Holland, 1987, 1988 など.
30) Azcorra, A.: Conversion Rules from ASN. 1 to ACT-ONE, SEDOS/C 1/WP/23/M (1987).
31) v. Bochmann, G. and Deslauriers, M.: Combining ASN. 1 Support with the Lotos Language, 9th IFIP WG 6.1, Int. Symp. on Protocol Specification, Testing, and Verification (1989).
32) Futatsugi, K., Goguen, J., Jouannaud, J.-P. and Meseguer, J.: Principles of OBJ 2, Proc. of 12th ACM Sympo. on POPL, pp. 52-66 (1985).
33) Futatsugi, K., Goguen, J., Meseguer, J. and Okada, K.: Parameterized Programming in OBJ 2, Proc. of 9th Int'l Conference on Software Engineering, pp. 51-60 (1987).
34) Goguen, J. and Winkler, T.: Introducing OBJ 3, Technical Report, SRI-CSL-88-9 (Aug. 1988).
35) Goguen, J. and Meseguer, J.: Order-sorted Algebra 1: Equational Deduction for Multiple Inheritance, Polymorphism, Overloading and Partial Operations, Technical Report, SRI-CSL-89-10 (1989).
36) Osterwil, L.: Software Processes are Software too, Proc. of 9th Int'l Conference on Software Engineering, pp. 2-16 (1987).
37) Katayama, T.: A Hierarchical and Functional Software Process Description and its Enaction, Proc. of 11th Int'l Conference on Software Engineering, pp. 343-352 (1989).
38) Ohmaki, K., Futatsugi, K. and Takahashi, K.: A Basic LOTOS Simulator in OBJ, Int'l Conference of Information Technology Commemorating the 30th Anniversary of the Information Processing Society of Japan (Info Japan '90), to be presented (Oct. 1990).
39) 例えば Steele, G. L. Jr. and Sussman, G. J.: Constraints, MIT AI Lab. Memo 502, M. I. T. Cambridge, Mass. (Nov. 1978).
40) ISO/IEC/JTC 1/SC 21/WG 7 N 109: Working Document on Topic 6.1—Modelling Techniques and Their Use in ODP (Apr. 1989).
41) 高橋, 中川路：構文定義用言語 ASN. 1 の特質と処理系の現状と動向, 情報処理, Vol. 31, No. 1, pp. 56-64 (1990).

(平成2年8月9日受付)