# 仕様記述言語ＴＥＬＬ／ＮＳＬによるプロトコルの仕様記述とその検証

槇本　肇・米崎　直樹・佐伯　元司・荒俣　博

東京工業大学　工学部

## １．はじめに

設計されたプロトコルに論理的矛盾がないか、製作された通信プログラムがプロトコルに従って動作するかといった性質を検証するためには、プロトコルの仕様が形式的に記述されていなければならない。また、検証を効率的に支援するためにも了解性の良い仕様記述言語が用いられるべきである。

これまでにも、各種の形式的仕様記述法が提案され、ペトリネットや状態遷移図に代表される状態遷移機械[6]が一般的に用いられている。しかし状態遷移機械に基づく仕様化技法では、対象システムが複雑になると状態数が飛躍的に増大する。さらに、システムの抽象化や階層化が十分に支援されていないこともその欠点として考えられる。抽象化や階層化は、大規模システムをわかりやすく記述するために、仕様記述言語にとっては不可欠の機能である。時間論理は仕様を記述したり、各種の検証を行なうのに適当であるが[3]～[5]、仕様記述言語としては、抽象化・階層化の機能を備えていないばかりでなく、習熟した人以外にはその内容が理解しにくい。

Tell/NSL[2]は、自然言語（英語）を用いた仕様記述言語であり、その意味的基礎を時間論理に置く。また語彙分割の手法を用いた抽象化・階層化[1]の機能を持っている。本報告では、Tell/NSLを用いたプロトコルの仕様化技法と検証例について述べる。[8]さらにプロトコルのように多層アーキテクチャを持つソフトウェアシステムの記述も、Tell/NSLにより自然に記述が行なえることを示す。

## ２．Tell/NSLによるプロトコルの仕様記述

### ２．１　仕様記述言語Tell/NSL[2]

仕様記述言語Tell/NSLは、曖昧性のない自然言語を用いた仕様記述言語である。本仕様化技法では、まず対象システムの内容を自然言語で説明し、そのときに用いた単語の意味を次々に自然言語文で詳細に定義していくことによって仕様を記述する。我々がシステムを自然言語で説明する場合、無意識のうちに語句を選択し、文章を構成するが、これらの語句には通常まとまった概念が対応している。それがソフトウェアモジュールに対応していると考えることにより、語句の意味を次々に定義していくことがソフトウェアを階層的に分割していくことに対応する。これによって、ソフトウェアの階層分割を意識することなしに、理解の構造に即した自然な分割が得られる。さらに読者は、仕様を一見しただけで、記述内容の概略を理解することができる。というのは、使用されている語句や文より、そのモジュールのイメージを漠然とではあるが、即座に言語理解として持つことができるからである。

Tell/NSLでは、仕様定義は自然言語の語句を定義する形式で行なわれ、その形式には、機能定義、動作定義、クラス定義、動的クラス定義の４つがある。これらは、各々関数型システム、プロセス、後の２つがオブジェクト（データ）のソフトウェアモジュールに対応する。表2-1 にこれらの語句定義と、定義できる語句の構文カテゴリや記述概念との対応を示す。また、Tell/NSLは、語句定義の汎用化を行なうための機能を持っており、これを使用することにより同様な定義を何度も行なうことを避けることができる。

表2-1　Tell/NSLの語句定義

| | 静的性質 | 動的性質 |
|---|---|---|
| クラス | クラス定義 | 動的クラス定義 |
| | 普通名詞<br>＋名詞　　語句間の<br>形容詞　関係記述<br>動詞（名詞、<br>形容詞化された） | 普通名詞<br>＋名詞　　動作の<br>形容詞　タイミング<br>動詞（一般） |
| 機能動作 | 名詞　　主語、<br>形容詞　目的語間の<br>動詞　　関係記述<br>（名詞、<br>形容詞化された） | 動詞（一般）<br>名詞　　動作の<br>タイミング |
| | 機能定義 | 動作定義 |

Tell/NSLで記述された仕様は、一階の線形時間論理式に変換され、厳密ではあるがごく自然な意味が割り当てられる。変換は、ユーザの語句定義から生成される規則とNSL（英文）の構文規則ごとに用意されている変換規則を用いる。NSL で使用できる英文は、単純な平叙文（関係代名詞や否定詞は使用できる）に限定されており、is、a 、will、not 、which 、until などの一般的な語句の意味は、あらかじめ定義されている。

### ２．２　Tell/NSLによる動的システムの仕様記述

並列処理システムやプロトコルなどの仕様記述では、入出力関係の記述だけでなく、同期や排他制御といった動作のタイミングの記述も必要である。Tell/NSLでは、タイミングの記述も語句定義の一部として記述する。このような動的な性質が本質的な意味となる語句は、transmitやreceive といった一般動詞のカテゴリであり、これらを定義するために、２つの定義形式が用意されている。１つは動作定義で、１つの動作を表わす語（一般動詞）を定義する。この定義はソフトウェアシステムでは、プロセスの仕様記述であると解釈される。

もう１つは動的クラス定義で、データオブジェクトの集合、すなわちクラスを表わす語句（lineやqueue など）を定義する。この語句は自然言語では、普通名詞のカテゴリに属する。このようなクラス定義では、クラスを表わす普通名詞だけではなく、それと密接に関連した語句も同時に定義する。例えば、図2-3 に示した動的クラス定義lineでは、lineのインスタンスの１つを表わす語句empty line、lineによって修飾される動詞sendやreadも同時に定義される。これらの語句は、そのクラスのデータに対する演算に対応すると解釈される。定義する語句が動詞のときは、動作のタイミングや動作干渉に関する制御といった動的な性質が動作定義と同様にその意味として記述される。この動的クラス定義は、システムが使用するデータ型を抽象データ型として定義することに対応するが、通常の抽

象データ型[10]に対して動的性質の仕様を追加している。[5] 動的クラスのインスタンスは、自然言語では固有名詞のカテゴリに属し、ソフトウェアシステムでは、共有資源と解釈される。我々は、動的システムを、主に動詞として定義される協調するプロセスとそれらによってアクセスされる共有資源（名詞として定義される）とに分けて記述する。以下では、プロトコルの仕様記述例を基に仕様化技法を述べる。

２．３　Tell/NSLによるプロトコルの仕様記述

１つの通信系の概念的モデルは、図2-1 のようになる。この図において、terminalのlineに対するインターラクション$s_1$、$r_1$、$s_2$、$r_2$ の列を規定するものがプロトコル仕様(protocol specification)で、userとprotocol machineとのインターラクション$S_1$ $R_1$、$S_2$、$R_2$ の列を規定するものがサービス仕様(service specification) である。[3],[6] サービス仕様は、protocol machineの入出力動作、プロトコル仕様はprotocol machineの内部動作に関する記述でもある。本稿では、プロトコル仕様だけに限定する。

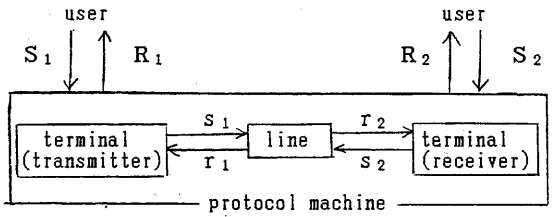


図2-1　通信システム

本仕様化技法では、そのようなインターラクションを互いに通信し合うプロセスの実行結果と見なし、回線lineを通信プロセスが使用する共有資源として記述する。従って、Tell/NSLでは通信プロセスは、動詞として動作定義によって定義され、回線は、動的クラス定義のインスタンスとして定義される。図2-1 中のterminalは、通信動作を表わす語句、例えばtransmit 、receive など、の定義の集合として記述される。

図2-2 に Alternating bit protocol(以下、AB-protocolと略す) と呼ばれる簡単な伝送誤り回復機能を持ったプロトコルのブロック図を、図2-3 にTell/NSLで記述した仕様を示す。AB-protocol に従って送信されるデータには、伝送シーケンス番号を表わす alternating bit と呼ばれる１bit のデータが付加される。Alternating bit は、１つのデータ伝送が正常に終了する度に反転される。送信側は、このようなデータをSRL lineを用いて受信側に送る。送信後は、受信側から応答(acknowledgement) が返ってくるまで待つ。受信側は、次に受信されるべきデータのalternating bitの値を記憶しており、データを受信すると直ちにalternating bit のチェックを行なう。異常が無ければ、正常に受信が終了する。受信側は、異常の有無にかかわらず受信したデータをそのままacknowledgementとしてRSL lineを介して送信側に送り返す。送信側は、タイムアウト時刻まで何も応答が無かったり、受信したacknowledgement のalternating bit が送信したデータのそれと異なっているときは、正しいacknowledgement が得られるまで同じデータを再送し続ける。これにより、回線上のデータ消失による伝送誤りを防ぐことができる。図中の SSN、RSN は各々送信側、受信側のalternating bit の値を記憶するためのカウンタであり、timer は送信側にタイムアウトを知らせるためのものである。

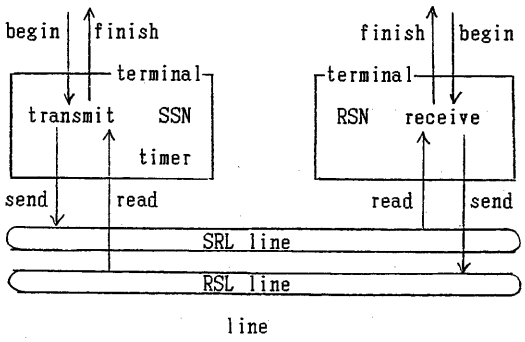


図2-2　AB-protocol システムのブロック図

図2-3 において、'AB-protocol machine'、'transmit'、'receive' といった語句の意味（仕様）が動作定義で定義され、'line'、'timer' といった語句が動的クラス定義で定義され、固有名詞SRL 、RSL はlineのインスタンスとして定義されている。インスタンスの宣言は、システムの構成を記述するconfiguration部でなされる。同様にalternating bit を記憶しておくためのカウンタSSN 、RSN が宣言されている。configuration 部の次の２つの文は、これらの初期状態を表わしている。

transmit、receive の定義のmeans that以下に続く箇条書き形式の文が各々の語句定義の本体、すなわち仕様の本体で、このプロトコルの詳細な内容をそのまま記述したものとなっている。文中に出現しているbegin 、finish、successfulといった不定詞や動名詞によって修飾されている語句は、各動作の進行状態を表わす語句である。

動的クラス定義 line では、sendやreadといった動詞も同時に定義され、それらの静的な関係(satisfy部で記述される）だけでなく、動作のタイミング（例えば、lineが空ならばread動作は空でなくなるまで待たされる、など）も記述されている。lineのsatisfy 部の文は、送信データが消失する可能性もあることを、timing部の3)の文は、何回もデータを送信すれば、そのうちのどれかは消失しないことを述べている。

予約語lexicon の後に続く部分は、すでに定義されている語句の意味をこの語句定義内でのみ使用できる新しい語句で表わすことを宣言する。

## ３．多層アーキテクチャシステムの仕様記述

通信システムは、ソフトウェアの変更のしやすさや汎用性という観点から多層アーキテクチャとなっている。多層アーキテクチャにおいては各層のシステムは隣接する上の層に対しサービスを提供し、それ自身は隣接する下の層のシステムによってサポートされている。その意味で、多層アーキテクチャはシステムの実現の階層構造である。多層アーキテクチャシステムの仕様を記述するためには、各層のシステムの仕様を記述するだけでなく、隣接する層との関係を記述しなければならない。

通信プロトコルの場合、どのように層を構成するかは標準化されており、その代表的な標準がISO のOSI

```
AB-protcol machine is the system such that
  Configuration
      1) There are line RSL and line SRL.
      2) There are counter SSN and counter RSN.
  Timing
  1) Initially RSL and SRL is empty.
  2) Initially RSN and SSN is 0.

  It transmits sequence of bit t means that
    1) Initially it begins to send the message made
       from sequence of bit t and SSN to line SRL.
    2) If it finishes sending to line SRL,
         then it begins to read from line RSL
           and timer is started.
    3) If it finishes reading acknowledgement A
         from line RSL and
      3-1) the alternating bit of A is SSN,
           then it begins to complement SSN.
      3-2) the alternating bit of A is not SSN,
           then it begins to send the message made from
             sequence of bit t and SSN to line SRL
             and timer is reset.
    4) If it finishes complementing SSN,
         then it finishes transmitting.
    5) If it doesn't finish reading from line RSL
         until timer is time-out,
         then it begins to send the message made from
           sequence of bit t and SSN to line SRL
           after timer is time-out.
  end  transmit ;

  It receives a sequence of bit means that
    1) Initially it begins to read from line SRL.
    2) If it finishes reading message m from line SRL and
      2-1) the alternating bit of m is RSN,
           then it begins to reply acknowledgement m
                                        to line RSL
             and it is successful in receiving
                 the data unit of m
                     until it finishes receiving.
      2-2) the alternating bit of m is not RSN,
           then it begins to reply acknowledgement m
                     to line RSL and
             it is not successful in receiving
             until it finishes replying to line RSL.
    3) If it finishes replying to line RSL and
       it is successful in receiving,
         then it begins to complement RSN.
    4) If it is successful in receiving sequence of bit
       t and it finishes complementing RSN,
         then it finishes receiving sequence of bit t.
    5) If it finishes replying to line RSL and
       it is not successful in receiving,
         then it begins to read from line SRL.

    Lexicon
      1) It replies acknowledgement A to line RSL
           := it sends message A to line RSL.
      2) It reads message m from line SRL
           := it receives message m from line SRL.

  end  receive ;

  Line associated with  send,receive, and empty line
    construction
      1)It sends message m to line l
           := the result of sending m to l is a line.
      2)It reads message m from line l
           := the result of reading from l is
                                  m and a line.
      3)Empty line is a line.
    Satisfy
      1) /read[send[m,l]] = <m,empty line> v
          send[m,l]=empty line/.
    Timing
      1) If it begins to read,
           then it is waiting to read
             until line is not empty.
      2) If it is waiting to read and line is not empty,
           then it is reading.
      3) If it begins to send infinitely often,
           then it will finish sending
                and then line is not empty.
  end line;

  Lexicon
    1) message := sequence of bit.
    2) acknowledgement := sequence of bit.
    3) Message m is made
           from sequence of bit t and bit b
                := m is the concatenation of b and t.
    4) The alternating bit of message m is bit b
                := b is  the head of m.
    5) The data unit of message m is sequence of bit t
                := t is the tail of m.
end  AB-protocol machine
```

図2-3　AB-protocol の仕様

reference model[7]である。２章でのAB-protocolは、OSI model の第２層(data link layer) に属するものである。プロトコルにおいては、その上下の層間の関係は、２章で示した各層のモデルを用いると、図3-1 のようにモデル化される。N層のN lineはN－１層のprotocol machineによって実現されている。このモデルでは、N層のlineに対する操作はN－１層のサービスを用いて行われる。この概念を明らかにするために、まず前節のAB-protocol の下層のプロトコルを考え、それらの関係を仕様化することを考える。

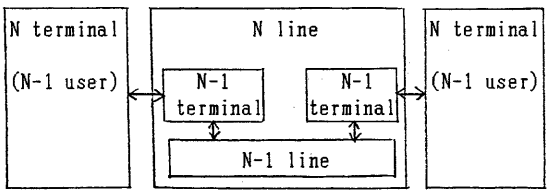


図3-1　層間の関係

３．１　Hardware level protocol

我々はここで考えるプロトコルをHardware level protocol と呼ぶことにする。これはクロックに同期してビット列を１ビットずつ伝送するものである。図3-2 にTell/NSLによる Hardware level protocolの仕様を示す。回線の値は０か１であり、データ転送中でないときは１である。送信側はまずスタートビット（０）を送信し、次にデータのビット列を送り、最後にストップビット（０１１１１）を送る。送信はクロックに同期して１ビットずつ行われる。また、受信側がデータ中のビット列をストップビットと誤って受理しないように、データ中に１が３つ以上連続するときは、まず最初の３つの１を送り、次に０を送信し、再び残りのデータの送信を始める。受信側はスタートビットを受信すると、ストップビットが送信されてくるまで送信側で挿入されたビット（０）を無視しながら受信を繰り返す。

３．２　層間のインターフェイス仕様

多層アーキテクチャシステムを設計するときには、各層の仕様のみでなく、層間のインターフェイスの仕様も決定しなければならない。インターフェイスの仕様は、その層のどの部分が、下の層のサービスによってどのように実現されているかを示したものである。Tell/NSLでは、このインターフェース仕様は上層の語句を意味的に等価な下層の語句の集合と対応づける形式で記述される。図3-3 にAB-protocol を第２層、Hardware level protocol を第１層としたときのインターフェイス仕様をTell/NSLによって記述した例を示

```
Hardware level protocol machine
                is the system such that
 Configuratuion
    1) There is line L.
    2) There is register Send-register.
    3) There is register Receive-register.
 Timing
  1) Initially L is 1.
  2) Initially it begins to receive.
  3) Initially it begins to transmit.
  4) Initially Send-register is cleared.
  5) Initially Receive-register is cleared.

  It transmit sequence of bit t means that
     configuration
       1) There is clock of baud-rate cycles Sclock.
     timing
      1) Initially it waits to transmit
           until Send-register is not empty.
      2) If it waits to transmit
         and Send-register is not empty,
           then it generate Sclock.
      3) If it finishes generating Sclock,
           then it is ready to transmit
                until Sclock is send-timing.
      4) If it is ready to transmit
         and Sclock is send-timing
           then it begins to put a start bit to L.
      5) If it finishes putting to L and it is not
         continuously transmitting 1 at three times and
        5-1) Send-register is not empty,
               then it begins to serial-output
                    from Send-register.
        5-2) Send-register is empty,
               then it begins to send stop bits.
      6) If it finishes serial-outputting bit b
           from Send-register,
         then it finishes serial-outputting bit b
              from Send-register
           until Sclock is send-timing.
      7) If it finishes serial outputting bit b
         from Send-register and it is not continuously
         transmitting 1 at three times and
         Sclock is send-timing,
           then it begins to put bit b to L.
      8) If it finishes putting to L and
         it is continuously transmitting 1
            at three times,
           then it is ready to insert bit 0
                      until Sclock is send-timing.
      9) If it is ready to insert bit 0
         and Sclock is send-timing,
           then it begins to put 0 to L.
     10) If it finishes sending stop bits,
           then it finishes transmitting.
     11) If it finishes transmitting,
           then (it waits to transmit
                    until Send-register is not empty)
             in the next time.


  It send stop bits means that
     1) Initially it is ready to send stop bits
                    until Sclock is send-timing.
     2) If it is ready to send stop bits and
        Sclock is send-timing,
           then  it begins to put 0 to L.
     3) If it is not continuously transmitting 1
        at four times and Sclock is send-timing,
           then  it begins to put bit 1 to L.
     4) If it is continuously transmtting 1
        at four times and Sclock is send-timing,
           then  it finishes sending stop bits.
 end send stop bits;

     It is continuously transmitting 1
          at i times   means that
       1) It finishes putting 1 to L at i times
                since it finishes putting 0 to L.
     end  continuously transmitting;

     Clock c is send-timing means that
       1)  c is 1.
     end  send-timing;
 end  transmit;

 It receives sequence of bit t means that
    state phrase
      1) ready to finish [adj].
  Timing
   1) Initially it waits to receive until
            a start bit is got from L.
   2) If it waits to receive and
      a start bit is got from L,
        then it begins to generate Rclock.
   3) If it finishes generating Rclock,
        then  it is ready to receive
           until Rclock is receive-timing.
   4) If it is ready to receive and
      bit b is synchronizingly got from L and
      it is not detecting stop bits
    4-1) it is continuously receiving 1
           at three times,
         then it stops receiving
              until Rclock is receive-timing again.
    4-2) it is not continuously receiving 1
           at three times,
         then it begins to serial-input b
                      to Receive-register.
   5) If it finishes serial-inputting
           to Receive-register
      and it is not detecting stop bits,
         then it is ready to receive
              until Rclock is receive-timing.
   6) If it stops receiving
      and Rclock is receive-timing,
         then it is ready to receive
              until Rclock is receive-timing again.
   7) If it is detecting stop bits,
         then it is ready to finish
             until it is continuously receiving 1
                   at four times.
   8) If it is ready to finish and
      it is continuously receiving 1 at four times,
         then  it finishes receiving.
   9) If it finishes receiving,
         then  (it waits to receive
                until a start bit is got from L)
           in the next time.

   It is detecting stop bits means that
     1) 0 is synchronizingly got from L.
     2) ((Rclock is receive-timing then 1 is got from L)
       until Rclock is receive-timing at four times)
                                    in the next time.
    end  detecting;

   Clock c is receive-timing means that
     1) c is 1.
   end  receive-timing;

   It is continuously receiving 1 at integer i times
      means that
       1) 1 is synchronizingly got from L at i times
            since 0 is synchronizingly got from L.
    end  continuously receiving;

   lexicon

     1) Bit b is synchronizingly got from line L
          := Rclock is receive-timing
             and b is got from L.
 end  receive;
```

図3-3 Hardware level protocol の仕様

```
Line  associated with put and got
      is implemented by bit.
  construction
    1) It puts bit b to line L
         := The result of putting b to L is a line.
  satisfy
    1) / put[L,b]=b /.
  Timing
    1) If it begins to put bit b,
         then it finishes putting bit b
              in the next time.

  Bit b is got from line L means that
    1) b is L.
 end got;

end  line;

Bit b is a start bit  means that
   1) b is 0.
end start bit;

Clock of i cycles  associated with generate
   construction
      1)It generates clock
          :=  The result of generating is a clock.
      2)It advances clock
          :=  The result of advancing is a clock.
     satisfy
       1) / gen[]=0 /.
       2) / advance[x]=mod[x+1,i] /.
     Timing
       1) If it begins to generate clock
            then (it finishes generating clock and
                  it begins to advance clock)
                  in the next time.
       2) If it begins to advance clock
            then it finishes advancing clock
                 in the next time.
       3) If it finishes advancing clock
          and it doesn't begin to generate clock
            then (it begins to advance clock) until
                 (it begins to generate clock
                  in the next time).
end  clock;
```

図3-3 Hardware level protocol の仕様（続き）

す。最初の文は、AB-protocol のlineがHardware level protocol によって実現されていることを表わしている。それに続く文は、AB-protocol 中に出現した語句とHardware level protocol 中に出現した語句の対応を記述したもので、例えば1)の文ではAB-protocol の 'l is empty' という文の意味はHardware level protocol中の語句を使用した文 'the register corresponding to l is cleared or it doesn't wait to receive'という意味と等価であるということが示されている。ここで、the register corresponding to l は、l に対応する下層でのregisterを指す語句である。

また、上の層のひとつの語句が、下の層のひとつの語句に必ずしも対応するとは限らない。この場合、上の層の語句の意味を語彙分割の手法を用いて下層の複数の語句に分解することになる。これはソフトウェアでは、下層の複数の機能をまとめて上層の１つの機能を実現するモジュールを組み立てたことに対応する。図3-3 では、pass down, pass upがそれぞれAB-protocolのlineのオペレーションのsend, readに対応するように定義されている。

## ４．時間論理式への翻訳

図2-3 や図3-2 で示したようなTell/NSLで記述された仕様は、時間論理式に翻訳されて厳密な意味が割り当てられる。翻訳は、文中に出現している語句の翻訳

```
Register
     associated with  clear,preset,parallel-load,
                      serial-input and serial-output
                 is implemented by sequence of bit
 construction
   1) It clears register R
        := the result of clearing is a register.
   2) It presets sequence of bit x to register R
        := the result of presetting x to R
              is a register.
   3) It parallel-loads sequence of bit x
        from register R
        := the result of parallel-loading from R
              is a sequence of bit and a register.
   4) It serial-inputs bit b to register R
        := the result of serial-inputting b to R
              is a register.
   5) It serial-outputs bit b from register R
        := the result of serial-outputting from R
              is a bit and a register.

 satisfy
   1) The result of clearing is empty sequence.
   2) The result of presetting x to R is x.
   3) The result of parallel-loading from R
        is R and empty sequence.
   4) The result of serial-inputting b to R
        is the concatenation of b and R.
   5) The result of serial-outputting from R
        is the head of R and the tail of R.

 timing
   1) If it begins to clear,
        then it finishes clearing in the next time.
   2) If it begins to preset,
        then it finishes presetting in the next time.
   3) If it begins to parallel-load,
        then it finishes parallel-loading
          in the next time.
   4) If it begins to serial-input,
        then it finishes serial-inputting
          in the next time.
   5) If it begins to serial-output,
        then it finishes serial-outputting
          in the next time.
end register
 lexicon
     1) Register r is empty := r is cleared.
end hardware level protocol machine
```

と構文に依存した翻訳規則とを用いて行なわれる。この翻訳手法は、モンテギュー文法[8] のそれを応用したものである。[1]

ここで使用する時間論理は、一階の線形時間多類論理で、時間オペレータとして、◇、○、until を持つ。直感的には、◇ＡはＡが将来いつか真になれば真、A until B は、Ｂが真になるまでＡが真であり続ければ真であることを表わす。これらは、各々助動詞will、副詞句in the next time、接続詞until の翻訳である。

複雑な仕様をわかりやすい自然言語で簡潔に記述するためには、これら以外の時間を表わす副詞や接続詞も必要である。これらの語句の意味は、マクロな時間オペレータや関数によって定義される。例えば図3-2 に使用されている until...at i times 、at i times since は、各々以下のように定義されたuntil-$_i$ 、times に翻訳される。

A until-$_i$ B は、Ｂがｉ回偽から真になるまでＡが真であり続ければ真、times[A,B]=iはＡが最近真になってからＢはｉ回真になったことを表わす。

$$A \text{ until-}_1 B \trianglelefteq A \text{ until } B$$

$$A \text{ until-}_{i+1} B \trianglelefteq A \text{ until } (B \wedge \sim\bigcirc B \wedge \bigcirc(A \text{ until-}_i B)) \quad \text{for } i \geqq 1$$

```
Line in AB-protocol machine is implemented
  by Hardware level protocol machine  means that
    1) l is empty
        := the register corresponding to l is cleared
           or it doesn't wait to receive.
    2) It sends message m to line l
        := it passes down message m.
    3) It reads message m from line l
        := it passes up message m.
    4) ll is the result of sending message m to l2
        := the register corresponding to ll is
              the result of presetting m to
                 the register corresponding to l2
           and it waits to receive.
    5) m and ll is the result of reading from l2
        := m and the register correspoding to ll
           is the result of parallel-loading
            from the register corresponding to l2.
    6) line := Receive-register.

   It passes down sequence of bit m  means that
       1) Initially it begins to preset m
          to Send-register and it is passing down.
       2) If it finishes presetting to Send-register,
           then it is ready to finish passing down
                 until it finishes transmitting.
       3) If it is ready to finish passing down
          and it finshes transmitting,
           then it finishes passing down
                in the next time.
   end pass down;

   It passes up sequence of bit m   means that
       1) Initially it waits to pass up
          until (Receive-register is not empty and
                 it waits to receiving).
       2) It waits to pass up and Receive-register
                is not empty and it waits to receiving.
            ↔ it is passing up.
       3) If it is passing up,
           then it begins to parallel-load
                 from Receive-register.
       4) If it finishes parallel-loading m
                 from Receive-register,
           then it finishes passing up m.
   end pass up;

   lexicon
     1) Register r is empty := r is cleared.
```

図3-3 AB-protocol とHardware level protocol のインターフェイス仕様

```
(～A ∧○A ∧○B)→○(times[A,B]=1)
(～A ∧○A ∧○～B)→○(times[A,B]=0)
(～B ∧○B)→
   (times[A,B]=x →○(times[A,B]=x+1))
 ○～B → (times[A,B]=x→○(times[A,B]=x))
(B ∧○B)→
   (times[A,B]=x →○(times[A,B]=x))
```

図4-1 、4-2 、4-3 にAB-protocol のline、Hardware level protocol のreceive とregister、インターフェース仕様の翻訳結果を示す。ここで□Aは、～◇～Aである。すなわちAが将来ずっと真ならばAが真である。動的クラスline、registerの翻訳は、スキーマ形式となっており、それらのインスタンスごとにスキーマから論理式が生成される。例えばregisterのインスタンスReceive-registerに対しては、図4-2 中のすべてのregister、演算名A の出現を各々Receive-register、A-Receive-registerで置き換えた式が生成され、この論理式が検証に使用される。インターフェース仕様の前半部は、検証の際に使用する変換スキーマとなっている。

```
Line
 (1) read[send[m,1]] = <m,empty-line> ∨
     send[m,1]≠empty-line
 (2) sending ∧ send-arg=a ∧ line=q
       → ○◇(finish(send) ∧ line=send[a,q])
 (3) reading ∧ line=q
       → ○◇(finish(read) ∧ line=arg2[read[q]]
                         ∧ read-arg=arg1[read[q]])
[timing part]
 (4) begin(read)
       → waiting(read) until line≠empty-line
 (5) waiting(read) ∧ line≠empty-line → receiving
 (6) begin(send) → sending
 (7) □(sending → ◇(finish(send) ∧ line≠empty-line)
```

図4-1　AB-protocol のlineの翻訳

## 5．レイヤ・プロトコルの実現性の検証例

Tell/NSLによって記述された仕様は、時間論理式に変換される。そのため、これらの時間論理式の集合をもとにしてシステムの動作の性質を検証することが可能である。システムの動作の性質には、safety property, liveness property, precedence propertyに分類され、それぞれの一般的な検証法も確立されている[4]。Tell/NSLでは、検証すべき性質も仕様と同様に階層的に記述することができ、その結果、検証すべき性質もわかりやすい記述となる。

多層アーキテクチャシステムの場合には、これらの性質以外にも、各層が下の層のサービスによって正しく実現されているかを検証する必要がある。これは実現性の検証の一種で、本報告では、この検証例について述べる。

Tell/NSLにおける実現性の検証は次の手順で行われる。

1)証明したい仕様中の文章を時間論理式に変換する。変換された時間論理式をAとする。

2)Aをインターフェイス仕様( 図3-3)の記述をもとに、下層の等価な時間論理式A' に変換する。

3)下層の仕様から翻訳されてできる時間論理式の集合を用いて、Init → □A'を証明する。ここでInitは下層のシステムの初期条件である。

以下でAB-protocol のlineの仕様( 図2-3)がHardware level protocol ( 図3-2)によって実現されているかを検証する。本報告で選んだ検証例は前提条件としてInitを使用しなくても証明できるため、簡単のために直接A'を証明した。これよりInit → □A'も容易に証明できる。

[検証例１] ：図4-1 の式1)

receive[send[m,l]]
  =<m,empty-line>∨send[m,l]=empty-line  ...1]

を証明する。Hardware level protocol では、回線の障害は起らないものとしているので、この式の∨の左半分

receive[send[m,l]]=<m,empty-line>  ...2]

が証明できる。この式は、

x=send[m,l]∧y=empty-line→ read[x]=<m, y> ..3]

と等価である。図4-3 の変換スキーマ1), 4), 5)を使用して、3]式に対応する式

x=preset[m,l] ∧wait(receive) ∧(y=clear[]∨
  ～wait(receive))→parallel-load[x]=<m,y> ...4]

を得る。この式は簡略化された式

x=preset[m,l] ∧wait(receive) ∧y=clear[]
   →parallel-load[x]=<m, y>  ...5]

```
receive
  1) begin(receive)
       → wait(receive)
         until ∃[got[L,x] ∧ startbit[x]]
  2) wait(receive) ∧ got[L,b] ∧ start-bit[b]
       → begin(generate-Rclock)
  3) finish(generate-Rclock)
       → ready(receive) until receive-timig[Rclock]
  4) ready(receive) ∧ synchronizingly-got[L,b]
     ∧ ~detecting-stop-bits →
     4-1) continuously-receive[it,3]
            → stop(receive)
              until_2 receive-timing[Rclock]
     4-2) ~continuously-receive[it,3]
            → begin(serial-input-Receive-register)
              ∧ serial-input-Receive-arg=b
  5) finish(serial-input-Receive-register)
     ∧ ~detecting-stop-bits
       → ready(receive) until receive-timing[Rclock]
  6) stop(receive) ∧ receive-timing[Rclock]
       → ready(receive)
         until_2 receive-timing[Rclock]
  7) detecting-stop-bits
       → ready-finish(receive)
         until continuously-receive[it,4]
  8) ready-finish(receive)
     ∧ continuously-receive[it,4]
       → finish(receive)
  9) finish(receive)
       → ○(wait(receive)
                until ∃[got[L,x] ∧startbit[x]])

  detecting-stop-bits
    1) synchronizingly-got[L,0]
    2) ○(receive-timing[rclock]→got[L,1])
         until_4 receive-timing[Rclock]

  receive-timing[c]
    1) c=1

  continuously-receive[it,i]
    1) times[synchronizingly-got[L,1],
             synchronizingly-got[L,0]]=i

  lexicon
    1) synchrinizingly-got[L,b]
       ↔ receive-timing[Rclock] ∧ got[L,b]

register
  1) clear[]=empty-sequence
  2) preset[x,r]=x
  3) parallel-load[r]=<r,empty-sequence>
  4) serial-input[b,r]=concat[b,r]
  5) serial-output[r]=<head[r],tail[r]>
  6) begin(clear)
       → ○◇(finish(clear) ∧ register=clear[])
  7) begin(preset) ∧ preset-arg=x ∧ register=r
       → ○◇(finish(preset) ∧ register=preset[x,r])
  8) begin(parallel-load) ∧ register=r
       → ○◇(finish(parallel-load)
            ∧ parallel-load-arg=arg1[parallel-load[r]]
            ∧ register=arg2[parallel-load[r]])
  9) begin(serial-input) ∧ serial-input-arg=b
     ∧ register=r
       → ○◇(finish(serial-input)
            ∧ register=serial-input[b,r])
 10) begin(serial-output) ∧ register=r
       → ○◇(finish(serial-output)
            ∧ serial-output-arg=arg1[serial-output[r]]
            ∧ register=arg2[serial-output[r]])
timing
  1) begin(clear) → ○finish(clear)
  2) begin(preset) → ○finish(preset)
  3) begin(parallel-load) → ○finish(parallel-load)
  4) begin(serial-input) → ○finish(serial-input)
  5) begin(serial-output) → ○finish(serial-output)
```

図4-2　Hardware level protocol の翻訳

```
Interface
 transformation schema
  1)  (l=empty-line)' = l'=clear[]
                              ∨ ~wait(receive)
  2-1) (p(send))' = p(pass-down)
  2-2) (send-arg=m)' = pass-down-arg=m
  3-1) (p(read))' = p(pass-up)
  3-2) (read-arg=m)' = pass-up-arg=m
  4)  (ll=send[m,12])' =
          ll'=preset[m,12']
                    ∧ wait(receive)
  5)  (<m,ll>=read[12])' =
          <m,ll'>=parallel-load[12']
  6) line' = Receive-register

 pass down
  1) begin(pass-down) ∧pass-down-arg=m
       → begin(preset-Send-register)
         ∧ preset-Send-register-arg=m ∧ passing-down
  2) finish(preset-Send-register)
       → ready-finish(pass-down)
         until finish(transmit)
  3) ready-finish(pass-down) ∧ finish(transmit)
       → ○finish(pass-down)

 pass up
  1) begin(pass-up)
       → wait(pass-up) until
         (~empty[Receive-register] ∧ wait(receive))
  2) (wait(pass-up) ∧ ~empty[Receive-register] ∧
      wait(receive)) ↔ passing-up
  3) passing-up
       → begin(parallel-load-Receive-register)
  4) finish(parallel-load-Receive-register)
       ∧ parallel-load-Receive-register-arg=m
         → finish(pass-up) ∧ pass-up-arg=m
  lexicon
    1) empty[r] ↔ clear[]=r
```

図4-3　インターフェース仕様の翻訳

と等価である。この式をHardware level protocol の仕様より証明する。まず図4-2 のregisterの仕様1)、2)、3)式より、

⊢x=preset[m,r] →parallel-load[x]=<m, clear[]>
⊢x=preset[m,r] ∧y=clear[]
　→parallel-load[x]=<m, y>　(等価変換)
⊢x=preset[m,r] ∧wait(receive) ∧y=clear[]
　→parallel-load[x]=<m, y>　(条件の付加)

よって 5]式が証明され、1]式が実現されていることが証明された。

[検証例２] ：図4-1 の3)式

```
reading ∧ line=q
  →○◇(finish(read) ∧ line=arg2[read[q]]
       read-arg=arg1[read[q]])            ...1]
```

lineのconstructor はempty,send,receiveしかないので、この３つの場合に分けて証明する。

• line=emptyのとき

1]式を図4-3 の変換スキーマ1)、3-1)、6)によって、下層の式に変換すると

```
passing-up∧(Receive-register=clear[] ∨
~wait(receive))
→○◇(finish(pass-up)
∧ pass-up-arg=
 arg1[parallel-load-Receive-register[clear[]]]
∧ Receive-register=
arg2[parallel-load-Receive-register[clear[]]])
                                        ...2]
```

この式を証明する。図4-3 のpass-up 仕様2)、lexicon より、

⊢～((passing-up∧Receive-register=clear[]) ∨ (passing-up∧～wait(receive))

⊢～Aより、⊢A→Bであるから、⊢2]である。

• line=send[m,l]のとき

同様にして下層の式に変換すると

passing-up∧Receive-register=c∧wait(receive)
→○◇(finish(pass-up)
∧ pass-up-arg=
arg1[parallel-load-Receive-register[c]]
∧ Receive-register=
arg2[parallel-load-Receive-register[x]])
where c=preset[m,Receive-register]
( 証明すべき式) ...3]

registerの仕様( 図4-2) 8)より

⊢parallel-loading ∧ register=x
→○◇(finish(parallel-load)
∧ parallel-load-arg=
arg1[parallel-load[register]]
∧ register=arg2[parallel-load[register]])
...4]

この式のregisterにReceive-registerを、変数xにpreset[m,Receive-register]をインスタンシエイトして、 pass upの仕様 3) 、4)式より

⊢passing-up ∧ Receive-register=c
→○◇(finish(pass-up)∧
pass-up-arg=
arg1[parallel-load-Receive-register[c]∧
Receive-register=
arg2[parallel-load-Receive-register[c]])
where c=preset[m,Receive-register] ...5]

を得る。これより条件部にwait(receive) を付加することにより、⊢3]

• line=arg2[read[q]]のとき

同様にして下層に変換すると

passing-up ∧ Receive-register=c
→○◇(finish(pass-up)
∧ pass-up-arg=
arg1[parallel-load-Receive-register[c]]
∧ Receive-register=
arg2[parallel-load-Receive-register[c]]))
where x=arg2[parallel-load[Receive-register]]
(証明すべき式) ...6]

この式は、registerの仕様8)を用いると line=send[m,l] のときと全く同様に証明できる。

⊢2]、⊢3]、⊢6]より1]式が実現されていることが証明された。

[検証例３] ：図4-1 の4)式

begin(read)
→ wait(read) until line ≒empty-line
...1]

図4-3 の変換スキーマ1)、3-1)を用いて

begin(pass-up)
→ wait(pass-up)
until (Receive-register≒clear[]
∧ wait(receive)) ...2]

となり、この式が証明すべき式である。ところが、pass up の仕様1)式は、lexicon を用いると、2]式になる。よって⊢2]である。

[検証例４] ：図4-1 の5)式

wait(read)∧ line≒empty-line
→ reading ...1]

下層の式に変換すると

(wait(pass-up)∧ Receive-register ≒clear[] ∧ wait(receive)) → passing-up ...2]

この式が証明すべき式であるが、pass up の仕様2)式は、lexicon を用いると、2]式になる。よって⊢2]である。

## ６．おわりに

本報告では、Tell/NSLを用いた多層アーキテクチャをした動的システムの仕様化技法を述べ、実際に２つの通信プロトコルの仕様を記述し、その実現性の検証を行なった。Tell/NSLは、自然なモジュール設計を提供し、読みやすく、かつ検証などの意味処理も可能である。今後の課題としては、半自動的に検証を行なうシステムの開発が上げられる。

謝辞：本研究に関し、熱心に議論して頂きましたＫＤＤ研究所の榑松明次長、浦野義頼室長、千葉和彦氏、滝塚孝志氏、および本学、榎本、米崎研究室の諸氏に感謝致します。

参考文献

[1] 佐伯・米崎・榎本：自然言語の語彙分割による形式的仕様記述、情報処理学会論文誌Vol.25、No.2、1984

[2] 榎本・米崎・佐伯：ソフトウェア開発システム(TELL)における自然言語による仕様記述言語(NSL) とその応用例、情報処理学会ソフトウェア工学研究会 34-14 、1984

[3] Schwartz,R.L. and Melliar-Smith,P.M.: From State Machines to Temporal Logic : Specification Methods for Protocol Standard, IEEE Trans. Commun., Vol.30, No.12 (1982) pp.2486-2496

[4] Manna,Z. and Pnueli,A. : Verifications of Concurrent Programs : The Temporal Framework, Correctness Problem in Computer Science, Academic Press (1981), pp.215-273

[5] Yonezaki,N. and Katayama,T. : Functional Specification of Synchronized Processes Based on Modal Logic, Proc.of 6th ICSE (1982), pp.208-217

[6] Sunshine,C.A. et al. : Specification and Verification of Communication Protocols in AFFIRM Using State Transition Models, IEEE Trans. Soft. Eng. Vol.8, No.5 (1982) pp.460-489

[7] Zimmermann,H. : OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection, IEEE Trans. Commun. Vol.28 (1980) pp.425-432

[8] 榎本・米崎・佐伯・荒俣：Tell/NSLによるレイヤ・プロトコルの仕様記述とその検証、第28回情報処理学会全国大会、1984

[9] Montague,R. : The Proper Treatment of Qualification in Ordinary English, Approaches to Natural Language, Reidel Dordrecht (1973)

[10] Guttag,J.V. et al. : Abstarct Data Types and Software Validation, Commun.ACM, Vol.21, No.12 (1978) pp.1048-1064