数理解析研究所講究録
596巻 1986年 277-292

待ち行列網解析ツールにおける モデル記述言語の一提案

渡辺　尚　(Takashi WATANABE)
中西　暉　(Hikaru NAKANISHI)
真田　英彦　(Hidehiko SANADA)
手塚　慶一　(Yoshikazu TEZUKA)

1. まえがき

近年、計算機網やローカルエリアネットワーク等の分散型システムが発達してきた。これらのシステムの性能評価を(近似)理論解析またはシミュレーションによって行おうとする待ち行列網解析ツールがＩＢＭをはじめ各所で開発されている[1]～[5]。これらのツールの開発主眼は、どのような理論解析法を行うかに絞られており、ユーザにとっての使いやすさを追及したシステムはあまり見られない。本研究では、待ち行列網解析ツールにおけるユーザインタフェースの向上を目的としたモデル記述言語を開発する。

現在のツールのモデル記述言語は、パラメータを指定する方法(パラメータ指定型)とモデルを構成する実体の様々な動作をプログラミングする方法(プログラミング型)の２種類に分けられる。パラメータ指定型言語は主に理論解析を行うツールに用いられている。この型の言語はパラメータによってモデル化が可能なものに対しては有効であるが、ツールが用意したパラメータ以外の性質は表現できない。また、プログラミング型言語はシミュレーションを実行するツール上で用いられており、様々なモデルを表現することが可能な反面、実体の性質のほとんどをプログラムの形で表現するため繁雑である。

従って、理論解析とシミュレーションの両方を行うようなツールに用いられるモデル記述言語としては、両者の長所を持ちかつ短所を補うような言語が望ましい。本稿では、既に提案しているシミュレーション専用言語ＳＩＬＱ[14][15] (SImulation language based on Logic for Queueuing network)について解説し、待ち行列網解析ツールにおけるモデル記述言語としてＳＩＬＱを検討する。

2. 待ち行列網シミュレーション専用論理型言語ＳＩＬＱ

ＳＩＬＱは待ち行列網シミュレーション(ＱＮＳ)ソフトウエアの主に記述性を向上させるために開発された言語であり、論理型言語Ｐrolog[6]を基礎としている。 Ｐrologは論理式をプログラムとみなす言語であり、プログラムの実行状態を考慮せずにプログラミングが可能であるため高い記述性及び検証性を有してい

るとされている。その反面、Prologで記述できる世界は静的な世界のため時刻という共通基盤をもつ複数のプロセスの関係の記述が不可能である。この欠点を補うためにT-Prolog[7]、Concurrent Prolog[8]、PARLOG[9]など時刻概念の表現方式がいくつか試みられてきた。しかし、これらの方法ではユーザがプロセス間の時刻関係を陽に記述しなければならないためPrologの利点を十分に生かしきれない。

そこでＳＩＬＱではプログラミングの対象をＱＮＳに限定しプロセス間の時刻関係をパラメータ化したタイムオブジェクトを用い、ユーザから複数のプロセスの時刻関係の陽な記述を解放している。

以下ＳＩＬＱの詳細について述べる。

３．ＳＩＬＱの基本設計方針

ＱＮＳ用言語の性能は、記述性、検証性及び経済性の３点から評価できると思われる。ＱＮＳ専用論理型言語ＳＩＬＱは高い記述性と検証性を実現するために以下の基本設計方針の元で開発した。

方針１．(言語の型)

イベントの発生条件を論理式によって記述し、ＱＮＳを定義する。

方針２．(陽な時刻関係記述の解放)

複数の時刻にわたる各プロセス間の関係(プロセス間の時刻関係)の陽な記述をユーザから開放する。

方針３．(ＱＮＳ用プリミティブ)

ＱＮＳを定義するのに必要なプログラミングプリミティブ(述語)を用意する。

方針４．(同種プロセスの一括定義)

ＱＮＳでは同種のプロセスが多数存在するモデルを対象とすることが多い。これに対し、本言語では同種プロセスの一括定義を可能とする。

方針５、(階層構造の表現)

ＱＮＳによく現れる階層的構造を容易に記述できる。

４．言語形式

現在シミュレーションに用いられている言語は待ち行列網モデルの記述の簡単化を目差した待ち行列網シミュレーション専用言語GPSSや処理速度が速い利点を持つFORTRANなどがある。しかし、これらでは複数のプロセス間の同期すなわちプロセスの時刻関係を陽に記述しなければならない。従って、FORTRANはもちろんGPSSでさえも対象とする網が大規模になりかつ複雑になるとプロセスの同期をとることが困難になってくる。

一方、ＱＮＳにおいては手続きによって"どのイベントの処理の後どのイベントが発生するか"を記述するよりも"あるイベントの生じる条件は何か"を記述した方が理解しやすいと考えられる。(例えば、待ち行列規律、分岐条件等がそれ

に相当する。）そこでＳＩＬＱでは、論理型言語を基礎とする。

論理型言語は現在いくつか提案されているが、その中でも、最も普及しており処理系も豊富なPrologを選択する。

ところで、Prologは一階述語論理に基づく言語であり、アルゴリズムを記述する手続き型言語と違って事物間の関係をホーン節によって記述する言語である。Prologによるプログラミングは論理プログラミングと呼ばれている。これは、通常の言語によるプログラミングがプログラムの実行状態すなわちアルゴリズムを考慮しなければならないのに対し、Prologでは事物間の関係の定義宣言がプログラムとなるためである。これをPrologがdeclarative semanticsを持つと言う。

この性質は、下に示すPrologの持つ２つの実行順序に関する非決定性に由来する。

①ルールの実行順序は予め決定されていない。

②ゴール文、またはサブゴール文中の述語の実行順序は予め決定されていない。

非決定性はどのルールや述語を先に実行しても結果が保証されると言い替えられる。即ち、すべてのルールが同時刻に処理可能であることを意味している。つまり、Prologで記述できる世界は同一時刻に発生するイベントのみからなる世界である。従って、ＱＮＳのように並行に動作するプロセスを内在しているジョブに対しては、複数のプロセスの記述が不可能であるため、そのままでは用いることができない。この点を解消するために、論理を基礎としたいくつかの分散処理システム記述用言語が提案されてきた(7)～(12)。これらの言語は、プロセス間の時刻関係記述の観点から次の３つに分類可能である。

① 時刻を直接取り入れる。すなわち、あるプロセスの内部状態が次に変化する時刻を記述する方法であり、この例としては、時相論理に基づく言語(10)(12)などが考えられる。

② メッセージ送受信、リソース束縛開放によってプロセス間の時刻関係を表現する。プロセス間の時刻関係はメッセージ送受信の際と、リソースの束縛、開放の際に生じると考える方法であり、この言語の例としては、Concurrent Prolog、PARLOGなどが考えられる。
①および②を両方取り入れた言語としてはT-Prolog、TS-Prolog(11)がある。

③ 他プロセスとの時刻関係がある決まった“型”で繰り返されることがわかっている場合にはその繰り返しを“型”として表現する。上記①②は、ユーザがプロセス間の時刻関係を陽に記述するのに対し、この方法はユーザが今考えているモデルがどのような“型”の組み合わせで実現できるかを考えることによって時刻関係を表現する。“型”をうまく設定すれば、ユーザに時刻関係記述の負担をおわすことがない。その一方、記述の対象は“型”作成の際に欠落する自由度のために限定される。

本稿で対象にするＱＮＳ向き言語について考察すると①②の方法では、結局プロセス間の時刻関係をユーザが記述するため、Prologのdeclarative semanticsを十分に生かしきれない。特に規模が大きく複雑な待ち行列網を扱った場合に、プロセス間同期の記述が煩わしくなり有効とは思われない。それに対して、③の方法では“型”を適切に選ぶことによってＱＮＳの記述性の向上が期待できるためこの方法を用いることにする。

５．タイムオブジェクト[15]

ＳＩＬＱではプロセス間の時刻関係の“型”としてタイムオブジェクトの概念を用いる。本稿で言うオブジェクトとは、最近急速に注目を浴びているオブジェクト指向プログラミングにおけるそれと同意である。一般にオブジェクトは以下の特徴を持つ。

１。各オブジェクトが独立に管理する内部変数を持つ。

２。外部からはメッセージを送ることによってのみオブジェクトの内部変数を操作可能である。

３。オブジェクトの状態は「メッセージを待っている状態(Ｗ状態)」と「処理を実行している状態(Ｅ状態)」のいずれかである。

４。オブジェクトはメッセージを受け取るとＥ状態になる。

これに対し、タイムオブジェクトは上記のオブジェクトに時刻概念を導入したものである。つまり、上記特徴１,２については同様であるが、３,４については次のように言い替えたものである。

3'。タイムオブジェクトの状態は「メッセージを待つあるいはある条件が満たされるのを待っている状態(Ｗ状態)」、「処理を実行している状態(Ｅ状態)」、「ある時刻が来るまで待っている状態(Ｈ状態)」のいずれかである。

4'。タイムオブジェクトはＷ状態からメッセージを受け取るか条件が満たされた時、または、Ｈ状態から所望の時間を経たときにＥ状態になる。

ここでタイムオブジェクトの分類を行う。(図１)

(１)　時刻消費による分類

まず、通常のオブジェクトと時刻に関係するオブジェクトに分類できる。前者をnon-time-consumingオブジェクト、後者をtime-consumingオブジェクトと呼ぶことにする。この名前は、後者は状態遷移のなかにＨ状態を含み、この状態においては何の処理も実行せずにただ時間を消費しているにすぎないことからつけられている。

(２)　入出力関係による分類

タイムオブジェクトはさらに入出力の数によっても分類できる。ここでは図１に示す５つを考える。この分類において、多入力多出力のオブジェクトは、多入力一出力、一入力多出力の２つのオブジェクトによって実現が可能であると考える。

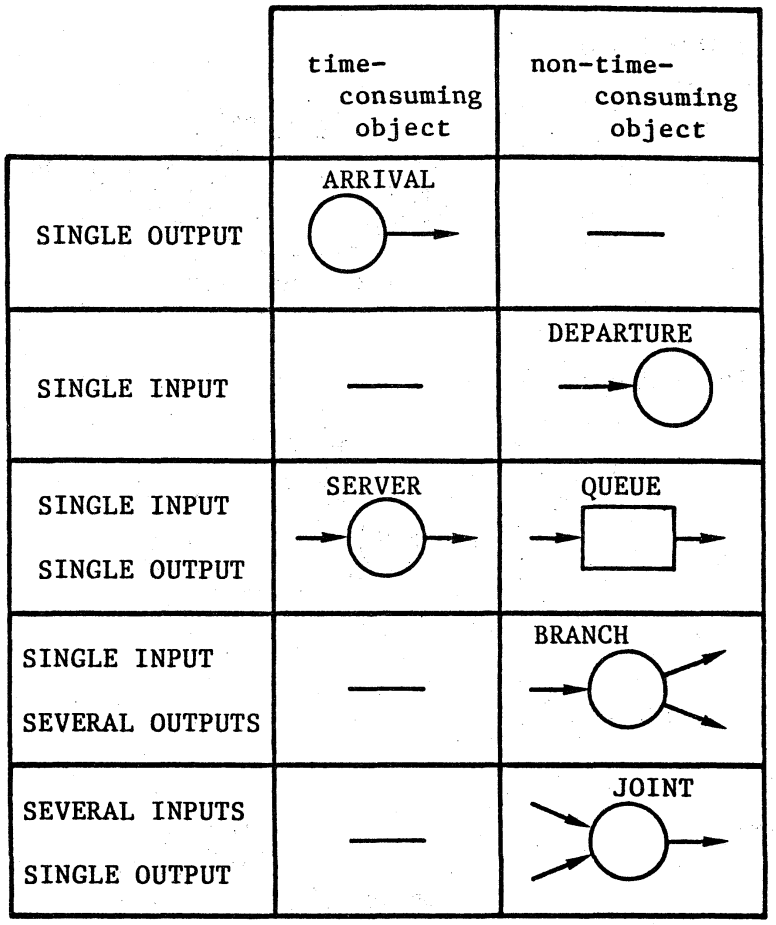


図１　タイムオブジェクトの分類

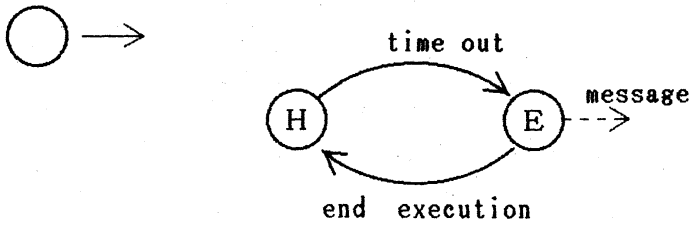
以上の２種類の分類法によって１０のタイムオブジェクトに分類される。

プロセス間の時刻関係は各プロセスがどのような状態を遷移するかに大きく関係している。上記の１０のタイムオブジェクトは一般的にはＥ,Ｈ,Ｗの３状態を任意に遷移するため、状態遷移の型としては無限個存在する。そのため、プログラミングの対象を一般化して考えると無限個のパターンが必要であるがＱＮＳに限定すると必要な状態遷移パターンは少なくなり、ＱＮＳに現われるプロセスは図１に示すような６つのタイムオブジェクトのいずれかに当てはまる。(*1)　この６つのオブジェクトを特にＱＮＳタイムオブジェクトと呼ぶ。各ＱＮＳタイムオブジェクトの状態遷移を図２に示し、以下にその説明を行う。

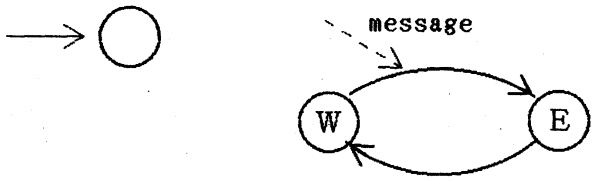
脚注*1　本研究で述べるＱＮＳタイムオブジェクトはクラスに、プロセスはインスタンスに相当する。

ただし、インヘリタンス及び実行中の生成、消滅については考慮していない。本言語ではソフトウエア記述の簡単化に的を絞っているため前者については必要性がないと思われる。後者については例えば計算機網におけるノードの障害等を対象とする場合には必要と思われるので、今後の課題とする。

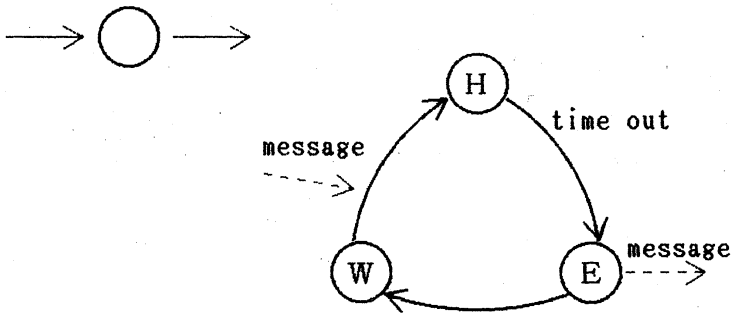
1 ARRIVAL



2 DEPARTURE



3 SERVER



4 QUEUE

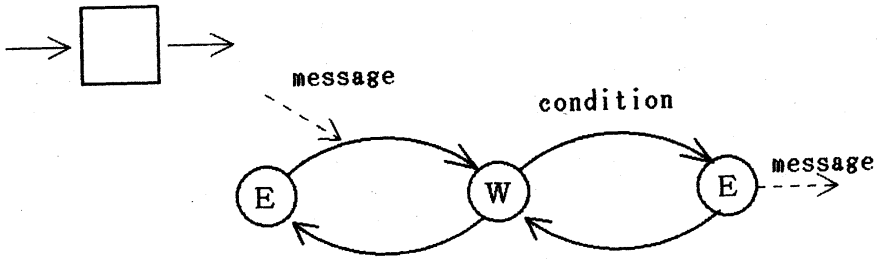
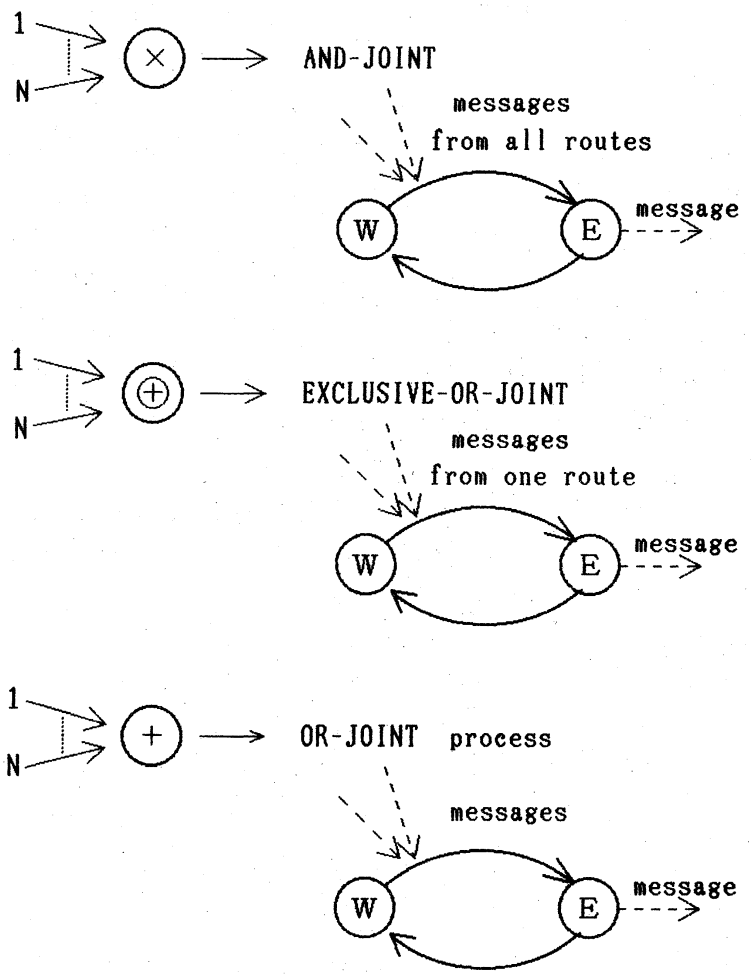


図2　各タイムオブジェクトの状態遷移図(その1)

5 JOINT



6 BRANCH

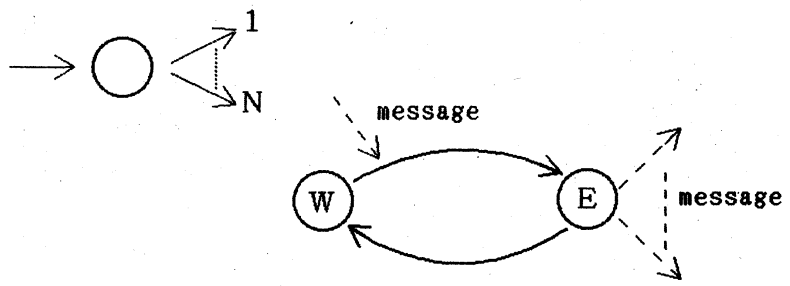


図2 (その2)

(i) ARRIVAL object

ARRIVAL objectは、まずある客に対する処理を行う(E状態)。その後、予め定

められた分布に従った時間だけ消費する。つまりこの時間がたつまでこのオブジェクトは休止している。これがH状態である。H状態からはtimeoutによって解除される。

(ii) DEPARTURE object

DEPARTURE objectは、受信したメッセージで表わされる客に対する処理(例えば統計収集など)を実行し(E状態)、その後次のメッセージを待つ。(W状態)

(iii) SERVER object

このobjectは、今受信したメッセージによって表わされる客に対して予め定められた確率分布に従った時間を消費する。ここでH状態となる。H状態を抜けたobjectはE状態となり客に対する処理を実行した後次のオブジェクトへメッセージを送信する。その後次のメッセージを待つ。

(iv) QUEUE object

QUEUE objectは、受信したメッセージを予め定義された規律に従って待ち行列に並べる。また待ち行列から出る条件が満たされると待ち行列から客を取り出し次のobjectに送り出す。その後次のメッセージを待つ。

(v) JOINT object

JOINT objectにはAND-JOINT object,OR-JOINT object及びXOR-JOINT objectの3つのタイプがある。これらのAND、OR、XORは単一時刻でのメッセージの送信についての論理である。すなわち、ある時刻にobjectがメッセージを送信すれば真、そうでなければ偽となる。(*2)

多入力のobjectは、2入力objectの拡張と考えられるため、以下では2入力objectについて示す。

(v－1)AND-JOINT objectはある時刻に objectの両方の入力経路にメッセージが受信されると新たなメッセージを送信する。もし、その時刻に1本でもメッセージが来ない経路があれば受け取ったメッセージは消滅する。

(v－2)XOR-JOINT objectはある時刻に2つの入力経路のうち片方だけから、メッセージを受信したときにメッセージを通過させる。

(v－3)OR-JOINT objectは、入力経路の内1本にメッセージを受信すれば直ちに次のobjectにメッセージを送る。(実際にはあまり意味を持たないがANDに対するJOINTとして加えておく。)

(vi) BRANCH object

このobjectでは受け取ったメッセージ以外にも自らメッセージを作り出すことができそれらを複数の出力経路から選択された経路を通して送信したり、メッセージを複写し放送型の通信を実現する。

脚注*2 ここでいう論理が論理学的に完全系をなすためには恒真のオブジェクトが存在しなければならない。このオブジェクトはシミュレーション時刻が更新されたときに常にメッセージを出力するものであり、これはBRANCH objectのメッセージ複写機能を用いて実現可能である。

６．ＱＮＳソフトウエアの構成要素とプログラミングプリミティブ

６．１　ＱＮＳソフトウエアの構成要素

ＳＩＬＱではＱＮＳソフトウエアの最小構成要素であるプログラミングプリミティブ(一種の組み込み述語)を提供している。まず、ＱＮＳソフトウエアに記述されるべき要素について検討する。

一般に分散型システムは構造と動作によって規定される[13]。同様にＱＮＳも網を通り抜ける客、網を構成するバッファやサーバ等の構造および客が各サーバやバッファ内でどのように扱われるか等の動作から構成される。しかし、シミュレーション特有なものとしてシミュレーションの制御、必要な統計量等がある。まとめるとＱＮＳソフトウエアには以下の各項が記述される。

(１)構造

①ソフトウエア上での客の構造　　②サーバ、バッファの接続関係

(２)動作

①バッファ内での客の並び方　　②バッファから客が出られる条件

③サーバ内で客の受けるサービス　④複数の経路の中の１つ以上の経路選択など

(３)制御

①シミュレーション開始時刻　　②シミュレーション終了時刻など

(４)出力

①必要な統計量　　②デバッグ情報

ＳＩＬＱではこれらの内、客の構造以外に対して論理型言語Prologに基づいた形式で記述する言語である。 客については以下に示すようにシステムフィールドとモデルフィールドからなるリストで表現される。

$$[[[T_1,T_2,\cdots,Tn],[[P_1,P_2,\cdots,Pn],[*******]]$$

System Field　　Model Field

システムフィールドは客が現在まで通ってきたプロセス名と出た時刻のリストの集合から成る。この履歴の管理はシステムが独自に行う。モデルフィールドはユーザが自由に書き込み読み出しができる。モデルフィールドによってユーザがより細かいレベルのシミュレーションが行えるようになっている。

６．２　プログラミングプリミティブ

ＳＩＬＱでは、ユーザが待ち行列網をより簡単に定義できるように、３０個のプリミティブを用意している。ユーザはこれらを用いて、待ち行列網の構造、動作、制御及び出力を別々に定義できる。３０個のプリミティブは、構造に関するプリミティブ１個、動作に関するプリミティブ１６個、制御に関するプリミティブ２個、統計量定義のプリミティブ１１個である。もちろん、ユーザは必要に応じてプリミティブ以外の述語の定義ができる。プリミティブのシンタクス及び、一覧表を付録１、２に示す。

[Example]

ここで対象とするシステムは図３(a)に示す３局直列網である。以下にシミュレーションの仮定を示す。

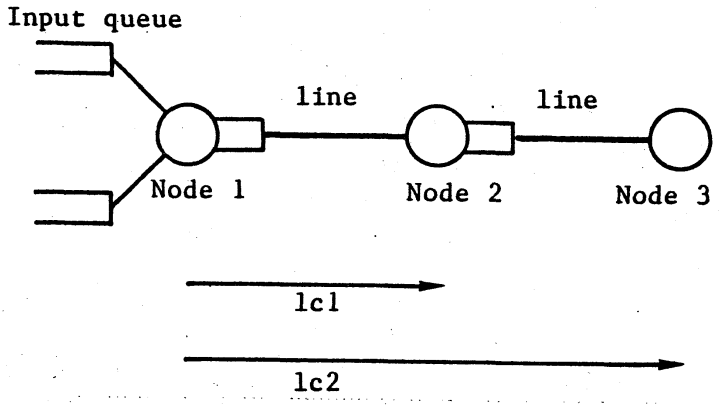
①論理チャネルlc１,lc２のウィンドウサイズはそれぞれ２と３である。

②パケットの発生間隔は各lcそれぞれ平均３秒と平均２秒の指数分布に従う。

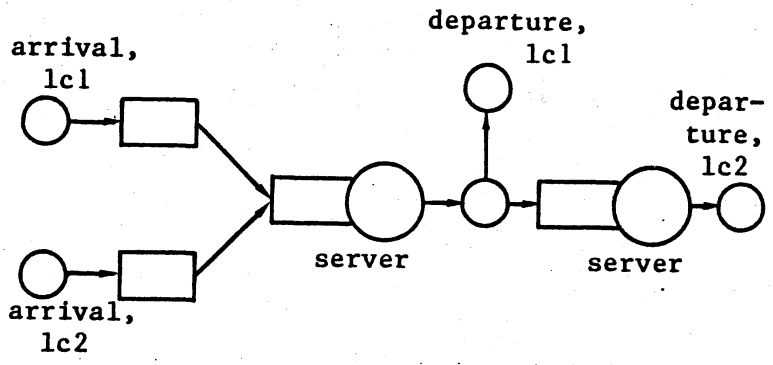
③一パケットを伝送するために要する時間は平均１秒の指数分布である。

④局内処理は無視できるとする。

このシステムを各回線の部分をサーバと見なし、タイムオブジェクトを用いてモデル化すると図３(b)のような２つのサーバを含むモデルとなる。ＳＩＬＱによるシミュレーションプログラムのうち動作及び構造の定義の部分を図４に示す。



(a) Three-node network simulation model



(b) Two-server model in Time-object

図３ウィンドウフロー制御シミュレーションモデル

同図に示すようにＳＩＬＱによるプログラミングはモデルの定義そのものである。例えば、パケット発生間隔については第３、４行目にプリミティブinterval を用いて定義されている。また、入力待ち行列[queue,lc1],[queue,lc2]から客が出られる条件は第７行～第１３行に定義されている。入力待ち行列から出た客数と退去した客数の差がウィンドウサイズ以下である場合にこの条件は満たされ

６．３　プリミティブの分類

プリミティブは、まず以下に示すシステムプリミティブとユーザプリミティブと呼ばれる２種類に大きく分けられる。

①　システムプリミティブは、システム内部で定義されている述語であり、ユーザはホーン節の本体部にのみ用いることができる。システムプリミティブの例としては、各プロセッサの処理履歴やプロセスの現在の時刻等がある。これらはPrologにおける組み込み述語に相当する。

②　ユーザプリミティブはユーザが定義する述語である。つまり、ユーザプリミティブはホーン節の頭部に用いる。

この他にプロセッサはどのようなプロセスの性質を決定するかによっても以下のような分類が可能である。

（１）　構造の定義　(STRUCTURE primitive)

構造に関するプリミティブとしては、connectがある。connectは２つのプロセスを接続するプリミティブである。このプリミティブの第一引数はconnectによって表される接続を示す指標である。接続指標によって２つのプロセスの１つのプロセスとみなし階層的に表すこともできる。この有用性は待ち行列網によくあらわれる階層的な構造、例えば１つのサーバと１つのキューをまとめてノードとするような構造を容易に扱えるところにある。

（２）　動作の定義　(BEHAVIOR primitive)

動作に関しては各プロセスごとに異なるプリミティブと共通なプリミティブを提供している。

①プロセス毎のプリミティブ(INDIVIDUAL primitive)

プロセス毎のプリミティブとしては、客が待ち行列から出られる条件やモデルフィールドの書き替え、待ち行列規律、分岐条件等のユーザプリミティブと現在待ち行列に並んでいる客数等を示すシステムプリミティブがある。

②各プロセスに共通なプリミティブ(COMMON primitive)

これらはプロセスに入ってきた客やこのプロセスがシミュレーションを開始してから現在までの履歴の参照等を行うためにある。

（３）　制御の定義(CONTROL primitive)

ＳＩＬＱでは、シミュレーション制御のうちシミュレーション開始時刻と終了時刻を定義する。

（４）　出力の定義(OUTPUT primitive)

待ち行列長、待ち時間、遅延時間及びサーバ稼動率についての平均と分散を出力するプリミティブが用意されている。

７．ＳＩＬＱによる記述例

ここではＳＩＬＱの記述例としてウィンドウフロー制御を行うパケット交換網のシミュレーションを挙げる。

```
/*                WINDOW FLOW                    */
/*  Arrival  */
interval(lc1,T):-exp_dis(3,T).
interval(lc2,T):-exp_dis(2,T).
append_model_field(Id,[SF],[SF,[Id]]).
/*  Queue  */
queue_out_condition(Id):-
        member_of(Id,[lc1,lc2]),
        sys_customers_record([queue,Id],X),
        sys_customers_record([departure,Id],Y),
        C is X - Y ,
        window_size(Id,MP),
        C < MP.
window_size(lc1,2).
window_size(lc2,3).
queue_out_condition(Id,Tc):-
        member_of(Id,[1,2]),
        sys_customers_in_server(Id,[]).
queue_out_selection(Id,[Packet|Rest],Packet).
buffer_capacity(Id,3):-
        member_of(Id,[lc1,lc2]).
/*  Server */
server_capacity(Id,1).
service_time(Id,T):-
        member_of(Id,[1,2]),exp_dis(1,T).
/* Branch  */
branch_selection(1,1):-
        sys_customer([X,[1]]).
branch_selection(1,2):-
        sys_customer([X,[2]]).
/*    Connect   */
connect(_,[arrival,lc1],[queue    ,lc1]).
connect(_,[queue  ,lc1],[queue    ,1]).
connect(_,[arrival,lc2],[queue    ,lc2]).
connect(_,[queue  ,lc2],[queue    ,1]).
connect(_,[queue  ,1],  [server   ,1]).
connect(_,[server ,1],  [branch   ,1]).
connect(1,[branch ,1],  [departure,lc1]).
connect(2,[branch ,1],  [queue    ,2]).
connect(_,[queue  ,2],  [server   ,2]).
connect(_,[server ,2],  [departure,lc2]).
```

図４　例題のプログラムリスト

る。また、各queueでの待ち行列規律はＦＩＦＯであることが、暗黙に定義されている。もし、ユーザがＦＩＦＯ以外の待ち行列をモデル化する場合にはプリミティブqueue_priorityを用いて定義できる。

ここで、図４を用いてＳＩＬＱの待ち行列網解析ツールにおけるモデル記述言語としての能力を評価すると、この例では、客の発生間隔、サービス時間等をパラメータ的に指定しており、また入力待ち行列から出られる条件をユーザがプログラミングしている。このようにＳＩＬＱはパラメータ指定型と、プログラミング型のモデル記述の両方の利点を持っていると考えることができる。

しかし、あくまでもＳＩＬＱはシミュレーション用言語であり、理論解析に必要な情報(例えば、あるサーバのサービス率に別の分布を仮定すれば理論解析が可能であるなど)を提供してはいない。従って、ＳＩＬＱを解析ツールのモデル

記述言語として用いるためにはＳＩＬＱプログラムの入力を会話的に行うインタフェースが必要であると考えられる。

８．まとめ

本稿では、先に提案したＱＮＳ専用論理型言語ＳＩＬＱについて詳述し、さらに待ち行列網解析ツールにおけるモデル記述言語としてこれを検討した。

モデル記述言語としてはパラメータ型言語の利点である簡潔さとプログラミング型言語の広い適用性を持つことが望ましい。さらに、プログラミング型の場合、手続き型言語のようにプロセス間の同期を陽に記述するよりも、各プロセス内で生じるイベントの発生条件を記述したほうがより理解しやすい。

近年注目を浴びている論理型言語Ｐrologを基本としたＳＩＬＱは、パラメータ指定型とプログラミング型の２つの特徴を持ち、さらにプログラミングの際にも宣言的記述が保たれていることからモデル記述言語として適していると考えることができる。

現在、ＳＩＬＱ処理系はパーソナルコンピュータ上の逐次処理Ｐrologを用いて構成されており実際に動作している[15]。

今後の課題としては、プロセスの生成消滅等のＳＩＬＱの言語的機能追加もさることながら、理論解析手法の情報をユーザとやり取りしながらモデル化を補助する会話的インタフェースの構築を上げることができる。

また、ＳＩＬＱで記述されたモデルは、解析の対象となるシステムの持つ構造的な並列性を的確に表現しているため、これを利用したＳＩＬＱ並行処理系の検討も行う予定である。

文　献

[1] 小島,米田,田中,春木:"蟻塚とその利用環境," 情処研報 Vol.86,No.12(1986.2)

[2] 木村:"QNA:Queueing Network Analyzerについて," ｵﾍﾟﾚｰｼｮﾝｽﾞﾘｻｰﾁ学会誌 (1984.4)

[3] D.Potier and M.Veran:"The Markovian Solver of QNAP2 and Examples," International Seminar on Computer Networking and Performance Evaluation, (1985.9)

[4] C.Sauer,E.MacNair and S.Salza:"A Language for Extended Queueing Network Models,"IBM J. Res. Develop Vol.24, No.6 (1980.11)

[5] 紀 ,守田,小林:"BCMP型待ち行列網による性能評価ﾂｰﾙQM-X," 情処第24回全国大会,p.271,(1982)

[6] W.F.Clocksin and C.S.Mellish:"Programming in Prolog,"Springer-Verlag (1981)

[7] I.Futo and J.Seredi:"A Discrete Simulation System Based on Artifi-

cial Intelligence Methods,"Simulation and Related Fields North-Holland, (1982)
[8] E.Y.Shapiro :" A subset of Concurrent Prolog and Its Interpreter," Technical Report TR003 ICOT
[9] K.L.Clark and S.Gregory: "PARLOG : A Parallel Logic Programming Language," Research Report Dept.of Computing Imperial College of Science & Technology (1983)
[10] 藤田、田中、元岡:"時相論理によるハードウエア仕様記述とPrologを用いたゲート回路の検証" 情報処理学会論文誌、25、2(1984)
[11] I.Futo and T.Gergely:"A Logical Approach to Simulation (TS-Prolog)," Adequate System Modeling,Wedde,H. Springer-Verlag Berlin (1983) pp.25-51
[12] Krithivasan Ramamrithan and Robert M.Keller : "Specification of Syn-chronizing Processes," IEEE trans. on software engineering Vol.SE-9.No. 6(1983)
[13] B.Chen and T.Y.Raymond : " Formal Specification and Verification of Distributed Systems," IEEE trans. on software engineering Vol. SE-9. No.6(1983)
[14] 渡辺、小林、山口、中西、真田、角所、手塚:"待ち行列網シミュレーション専用論理型言語について" 情処学会マルチメディア通信と分散処理 26-3 (1985)
[15] 渡辺、中西、真田、手塚:待ち行列網シミュレーション専用論理型言語の処理系について" 昭和61年度信学会総全大S14-2(1986)

```
<statement>::=
  <unit cls>|<non-unit cls>|<query>
<unit cls>::=<head>.
<non-unit cls>::=<head>:-<goal sqnc>.
<head>::=<User primitive>|<Prolog term>
<goal sqnc>::=<goal>|<goal>,<goal sqnc>
<goal>::=<System primitive>|<Prolog term>
<query>::=sim(<file name>).
<User primitive>::=
  connect(<C_Id>,<P_Id>,<P_Id>)
 |append_model_field(<Ar_Id>,<Sys_Cstm>,<Cstm>)
 |interval(<Ar_Id>,<Time>)
 |queue_out_condition(<Q_Id>)
 |buffer_capacity(<Q_Id>,<Number>)
 |queue_priority(<Q_Id>,<Cstm>,<Cstm>)
 |server_capacity(<S_Id>,<Number>)
 |service_time(<S_Id>,<Time>)
 |customer_merge(<J_Id>,<List>,<List>,<List>)
 |branch_selection(<B_Id>,<Cstm>,<List>)
 |customer_change(<Cstm>,<Cstm>)
 |start_time(<Time>)
 |end_time(<Time>)
 |data_collect_section(<Time>,<Time>)
<System primitive>::=
  sys_customers_in_queue(<Q_Id>,<Cstms>)
 |sys_customers_in_server(<S_Id>,<Cstms>)
 |sys_time(<Time>)
 |sys_customers_record(<P_Id>,<Cstms>)
 |sys_customers(<Cstms> )
 |sys_change(<Prolog term>,<Prolog term>)
 |sys_customers_number(<P_Id>,<List>)
 |sys_mean_delay_time(<P_Id>,<P_Id>)
 |sys_div_delay_time(<P_Id>,<P_Id>)
 |sys_mean_queue_length(<Q_Id>)
 |sys_div_queue_length(<Q_Id>)
 |sys_max_queue_length(<Q_Id>)
 |sys_mean_queueing_time(<Q_Id>)
 |sys_div_queueing_time(<Q_Id>)
 |sys_max_queueing_time(<Q_Id>)
 |sys_utilization(<S_Id>)
<C_Id>::=<atom>|<C_Id>
<P_Id>::=
  <Ar_Id>|<Q_Id>|<S_Id>|<J_Id>|<B_Id>|<D_Id>
<Ar_Id>::=[arrival,<Id>]
<Q_Id>::=[queue,<Id>]
<S_Id>::=[server,<Id>]
<J_Id>::=[joint,<Id>]
<B_Id>::=[branch,<Id>]
<D_Id>::=[departure,<Id>]
<Cstm>::=[[<Sys_Cstm>]|<List>]
<Sys_Cstm>::=[]|[[<Time>,<P_Id>],<Sys_Cstm>]
<Cstms>::=[]|[<Cstm>,<Cstms>]
```

付録1　BNFによるSILQシンタクス

① STRUCTURE primitive

connect(C_Id,Ps_Id,Pr_Id)	C_Id:接続識別子
	Pr_Id:受信プロセス識別子
	Ps_Id:送信プロセス識別子

② BEHAVIOR primitive

INDIVIDUAL primitive

ARRIVAL	append_model_field(Id,Sys_customer,Customer)	
		Id:プロセスの識別子
		Sys_customer:システムフィールドのみの客
		Customer:モデルフィールド付きの客
	interval(Id,T)	メッセージ発生時間間隔T
QUEUE	queue_out_condition(Id)	待ち行列から客が出られる条件
	buffer_capacity(Id,C)	バッファの容量C
	queue_priority(Id,Hp,Lp)	待ち行列に並ぶ優先権の高い客(Hp)低い客(Lp)
	sys_customers_in_queue(Id,List)	待ち行列に並んでいる客(List)
SERVER	server_capacity(Id,C)	同時に同じサービスを受けられる客の最大数C
	service_time(Id,Ts)	サービス時間Ts
	sys_customers_in_server(Id,List)	現在サービスを受けている客のリスト
JOINT	customer_merge(Id,Customer1,Customer2,Customer3)	
		Customer1とCustomer2を結合した客Customer3
BRANCH	branch_selection(Id,List)	List:選択された経路のリスト

COMMON primitive

sys_time(T)	現在の時刻T
sys_customers_record(Id,List)	プロセスIdを出ていった客のリスト
sys_customers(List)	現在プロセスにいる客のリスト
sys_change(User(Pre),User(Post))	User(Pre):ユーザの定義した述語Userの前の状態
	User(Post):Userの後の状態
customer_change(Pre_customer,Post_customer)	Pre_customer:変化する前の客
	Post_customer:変化した後の客
sys_customers_number(Id,List)	プロセスIdを出ていった客の数

③ CONTROL primitive

start_time(St)	St:シミュレーション開始時刻
end_time(Et)	Et:シミュレーション終了時刻

④ OUTPUT primitive

data_collection_section(Cst,Cet)	Cst:データ収集開始時刻
	Cet:データ収集終了時刻
sys_mean_delay_time(Id,Id)	遅延時間の平均値の出力
sys_div_delay_time(Id,Id)	〃 分散の出力
sys_mean_queue_length(Id)	待ち行列長の平均値の出力
sys_div_queue_length(Id)	〃 分散の出力
sys_max_queue_length(Id)	〃 の最大値の出力
sys_mean_queueing_time(Id)	待ち時間の平均の出力
sys_div_queueing_time(Id)	〃 分散の出力
sys_max_queueing_time(Id)	〃 最大値の出力
sys_utilization(Id)	サーバの利用率

ただし、大文字で始まるものは変数

付録２　ＳＩＬＱプリミティブ一覧表