プログラミング言語　12−1
（１９８７・７・２４）

離散型シミュレーション言語ＳＩＬＱの処理系の構成

巽　秀宜[†]　渡辺　尚[††]　井上　健[†]　中西　暉[†]　手塚　慶一[†]
[†]　大阪大学　工学部　通信工学科
[††]　徳島大学　工学部　情報工学科

待ち行列網シミュレーション専用論理型言語ＳＩＬＱの処理系の開発について報告する。
ＳＩＬＱでは、時刻の進行概念を持つオブジェクトであるタイムオブジェクトを用いて待ち行列網モデルの構造が決定され、加えて、タイムオブジェクトのインスタンスであるＱＮプロセス相互間、及び網内を移動する客との論理的関係をＰｒｏｌｏｇ形式により記述することによって、シミュレーションソフトウェアが構成される。その処理系として、ＱＮプロセスの実行順序を制御するシミュレーション実行モジュール、ＱＮプロセスを処理するタイムオブジェクト実行モジュール、Ｐｒｏｌｏｇ形式のユーザ・プログラムに対する処理を行うホーン節処理モジュールから成るＳＩＬＱ処理系を設計し、Ｃ言語を用いて実現している。本処理系の設計の詳細について論じ、幾つかの実行例を通じて実用的な処理性能を持つことを明らかにする。

CONSTRUCTION OF A DISCRETE EVENT SIMULATOR FOR SILQ

Hidenori TATSUMI[†] , Takashi WATANABE[††] , Takeshi INOUE[†] , Hikaru NAKANISHI[†] and Yoshikazu TEZUKA[†]

[†] Faculty of Engineering, OSAKA University,
2-1, Yamadaoka, Suita, Osaka, 565, Japan
[††] Faculty of Engineering, Tokushima University

A queueing network simulator executed by using SILQ programs (SImulation language based on Logic for Queueing networks) is proposed. In the SILQ, simulation models are described by time-objects and horn-clauses including preassigned / user-defined primitives.

This simulator consists of timing engine based on the event driven control scheme, stochastic data collection module, and SILQ emulator (time-object modules and horn-clause processing module). This system, written in C language for the purpose of portabilities and efficiencies, has been implemented on 2 different types of computers. Some experiments based upon its validity and practical performance are discussed. Finally, some perspectives on the improvement are also shown.

１．はじめに

情報処理システムの高度化，複雑化に伴い、設計・管理の諸段階においてシステムの性能評価の役割が益々重要になりつつある。それらの中で通信網，計算機網，分散型・並行処理型情報処理システムなどのネットワークシステムを対象とした性能評価に関しては、待ち行列網モデルを用いた解析が有効な手段として用いられる。例えば、システム内のチャンネル及び計算機資源をサービス施設と見なし、メモリバッファを待ち施設に対応づけることにより、システムの性能指標である応答時間、信頼度、処理効率等を待ち行列網モデルの待ち時間、スループット等の特性量で評価することが可能になるなど、妥当かつ理解しやすいモデル化が可能である。

待ち行列網モデルの解析手段としては数学解析[1]とシミュレーションによる解析[2]-[4]が代表的である。このうち、数学解析は現状では適用できるモデルが限定されており、複雑なモデルに対しては近似評価が行われるのが一般的である。これに比べてシミュレーションは、その実行に多大な計算機資源を必要とはするが、より現実に近い状況での特性を評価することができ、近似解析の補完的手段として用いられている。近年では、シミュレーションは、システム動作の正当性の検証手段としても用いられるようになり、性能評価手法としての重要性が更に高まりつつある。

筆者らは、このような現状を踏まえ、シミュレーションソフトウェアの記述、及び検証の効率化を目的とした待ち行列網シミュレーション専用言語 ＳＩＬＱ (Simulation Languagebased on Logic for Queueing network)を提案している[5]-[7]。ＳＩＬＱは論理型言語Ｐrolog[8]を記述形式として持ち、宣言的に対象モデルを記述できるのが特徴であり、その有効性は種々の記述例を通して確認されている。

本稿では、ＳＩＬＱをプログラム言語として持つシミュレータの開発とその効率化を試みる。特に本シミュレータの実現には、ＳＩＬＱの持つ記述性に関する特徴を損なうことなく、高速にシミュレーションを実行する機構の開発が必要である。そこで、論理計算の必要な部分のみに導出処理を行うシミュレータを提案するとともに、本システムをシミュレーションの実効処理速度の面から評価する。

２．ＳＩＬＱの意義

待ち行列網シミュレーションにおいては、まず、ユーザによるシミュレーションソフトウェアの作成効率が問題となる。従来、待ち行列網モデルの構築時並びにシミュレーションソフトウェアの作成時におけるユーザの負担を軽減するために種々のシミュレーション専用言語が開発されてきた。代表的な言語として、ＧＰＳＳ[9]、ＳＬＡＭ[10]、ＳＩＭＳＣＲＩＰＴ[11]、ＲＥＳＱ[12]等が挙げられる。これらのシミュレーション専用言語には、シミュレーションモデルの構成及び動作の記述のために、実システムと対応の取りやすい典型的なモデル構成要素群が提供されており、また、シミュレーションの実行のためには、時刻管理機構をはじめとする制御機構が用意されている。

しかしながら、通信プロトコルなどの複雑なシステム機構を持つモデルの存在を考えれば、より簡潔にモデルを表現できる記述言語の開発がソフトウェアの生産性向上の要件であると考えられる。

待ち行列網モデルは、事象の生起条件をはじめとしてモデルの構成要素間の論理的な相互関係によって動作が決定される場合が多く、論理式の形でモデルを記述することが有利である。シミュレーション専用言語ＳＩＬＱは、論理型言語として代表的なＰrolog の記述形式を用いることにより、ホーン節によるモデルの仕様記述を可能にしている。以下では、ＳＩＬＱの記述形式に関する特徴について検討する。

２．１　タイムオブジェクトによるモデルの構築

タイムオブジェクトは、オブジェクト指向プログラミングの概念に基づく抽象化データであり、オブジェクト内部での論理時刻の経過を考慮したものである。待ち行列網モデルにおける客の移動は、タイムオブジェクトでは客を表すメッセージの受渡しと考える。ＳＩＬＱでは、待ち行列網モデルの構成要素として Arrival， Queue， Server， Branch， Joint， Departure の６種類のタイムオブジェクトを提供している。ＳＩＬＱにおいて、モデルを構成するプロセスはタイムオブジェクトのインスタンスであり、これらをＱＮプロセスと呼ぶ。図１にタイムオブジェクトとＱＮプロセスの関係を示す。

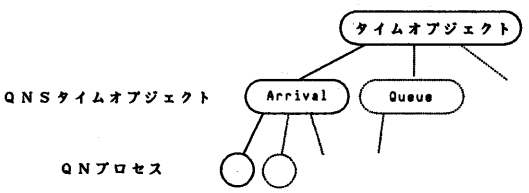

図１　タイムオブジェクトとＱＮプロセス

ＳＩＬＱでは、ＱＮプロセスを組み合わせることによって待ち行列網モデルの構造が表現される。以下に６種のタイムオブジェクトに属するＱＮプロセスの基本動作を簡単に述べる。

(1)Arrival プロセス

ある確率分布に従う時間間隔で客（メッセージ）を生成し、送出する。

(2)Queue プロセス

到着した客を待ち行列規律に従って格納施設（バッファ）に並べる。決められた条件が満たされればバッファ内の先頭の客を選び出し、その客（メッセージ）を送出する。

(3)Server プロセス

一定数の客を保持し、ある決められた時間だけ保留（サービス）する。その後、客（メッセージ）を送出する。

(4)Branch プロセス

出力経路の選択をする機能を持つ。複数の出力経路の幾つかを選び、同時にそれらへ客（メッセージ）を送出する。

(5)Joint プロセス

複数の入力経路からの客（メッセージ）をマージする機能を持つ。論理関係に従って、 And-Joint, Xor-Joint, Or-Joint の３つのタイプが用意されている。ただし入力経路は２本が基本であり、２本以上の入力経路を持つプロセスはこれらの組合せによって構成する。

(6)Departure プロセス

到着した客（メッセージ）を消去する。

これらのＱＮプロセスを用いてモデルの構築を行えば、モデルを構成する個々のプロセスを直感的に理解しやすいだけでなく、ＱＮプロセスの接続関係からモデルの構造を視覚的に把握することができる。

２．２　ホーン節を用いた条件判断と動作の記述

ＳＩＬＱでは、ＱＮプロセスの動作の記述、及びＱＮプロセス相互間の関連の記述等を行う手段として、 仕様をProlog 形式で記述するための述語（プログラミングプリミティブ）を提供している（表１）。これは、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に分けられる。

※ユーザプリミティブ　　１０個

ホーン節の頭部で用いるプリミティブである。ユーザは述語中の変数及び本体部を記述することによって、各ＱＮプロセスの動作内容の詳細とその適用条件の定義を行う。

※システムプリミティブ　５個

ホーン節の本体部で用いるプリミティブである。バッファに滞在中の客、通過客数といった各ＱＮプロセスの示す状態を表している。

必要に応じて、これらのプリミティブ以外に新しい述語をホーン節形式で定義することも可能である。また、各ＱＮプロセスは属するタイムオブジェクトの種類と識別子によって表現されるが、識別子を変数として与えることで複数のＱＮプロセスに対して共通の機能を一括して定義できる。この結果大規模なモデルに対してもプログラムの量を抑えることができる。

この他ＳＩＬＱには、シミュレーションの開始、終了時刻及び出力統計量の指定を行うプリミティブ等が用意され、シミュレーションソフトウェア全体がProlog 形式で記述される。従って、シミュレータにおいては、ユーザによって記述されたホーン節形式のプログラムを評価し、論理的処理を実現するための処理系を持つ必要がある。

これまで、ホーン節形式に時刻概念を導入したT-Prolog[13]等の言語では、時刻概念に関連する条件の表現が繁雑となり、記述性の低下を招く原因となってきた。これは、例えば一つのステートメント上で複数の時刻にまたがった状況を表現するには、手続き的な記述に頼らざるを得ないからである。ＳＩＬＱはタイムオブジェクトの内部でのみ時刻を扱うことによって、ユーザの記述レベルから時刻の概念を隠蔽している。 例えば、 プロセス内部で生じる事象の順序、あるいは、並行して動作するプロセス間の同期については、ユーザが記述する必要はない。この結果、

表１　プログラミングプリミティブ

USER primitives		
Arrival	append_model_field	：モデルフィールドへの記入
	interval	：客の到着時間間隔
Queue	queue_out_condition	：待ち行列から客を出す条件
	buffer_capacity	：待ち行列の最大容量
	queue_priority	：プライオリティ
Server	server_capacity	：Serverが同時にサービスできる客の最大容量
	service_time	：サービス時間
Joint	customer_merge	：客の合成
Branch	branch_selection	：客の送出経路の選択
共通	customer_change	：客のモデルフィールドの変更

SYSTEM primitives	
sys_customers_in_queue	：待ち行列に並んでいる客
sys_customers_in_server	：Server内でサービスを受けている客
sys_time	：現在の論理時刻
sys_customers_record	：プロセスを通過した客の総数
sys_customers	：プロセスに現在いる客

STRUCTURE primitives	
connect	：構造の定義

CONTROL primitives	
start_time	：開始時刻の設定
end_time	：終了時刻の設定

STATICAL primitives	
data_collect_section	：統計データ収集区間の設定
sys_mean_queue_length	：平均待ち行列長
sys_div_queue_length	：待ち行列長の分散
sys_max_queue_length	：最大待ち行列長
sys_mean_queueing_time	：平均待ち時間
sys_div_queueing_time	：待ち時間の分散
sys_max_queue_ing_time	：最大待ち時間
sys_utilization	：Serverの利用率
sys_customers_number	：プロセスを通過した客の総数
sys_mean_delay_time	：平均遅延時間
sys_div_delay_time	：遅延時間の分散

ＳＩＬＱを用いてモデルの仕様を記述する場合、それぞれのステートメントは時刻によらない静的な状況だけが表される。従って、シミュレータには、言語レベルで隠蔽された時刻管理をユーザと独立に実現する必要がある。

２．３　ＳＩＬＱによるモデルの記述例

ＳＩＬＱによる記述の一例として、図２に示すウインドウ・フロー制御を実装した２サーバ直列網モデルを取り上げる。

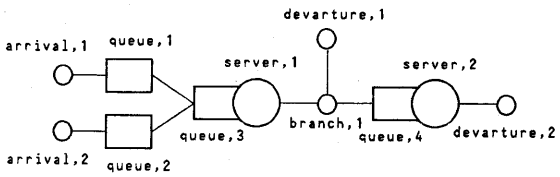

図２　２サーバ直列網モデル

```
interval(1,T):-exp_dis(30,T).
interval(2,T):-exp_dis(20,T).
append_model_field(X,X).

buffer_capacity(1,3).
buffer_capacity(2,3).
buffer_capacity(3,100).
buffer_capacity(4,100).

queue_out_condition(I):-member_of(I,[1,2]),
    sys_customers_record([queue,I],X),
    sys_customers_record([departure,I],Y),
    X=Y+C,
    window_size(I,M),
    C<M.

window_size(1,2).
window_size(2,3).

queue_out_condition(3):-
    sys_customers_in_server(1,[]).
queue_out_condition(4):-
    sys_customers_in_server(2,[]).

service_time(X,T):-exp_dis(10,T).
server_capacity(X,1).

branch_selection(1,7):-
    sys_customers([branch,1],[X,[1]]).
branch_selection(1,8):-
    sys_customers([branch,1],[X,[2]]).

connect(1,[arrival,1],[queue,1]).
connect(2,[queue,1],[queue,3]).
connect(3,[arrival,2],[queue,2]).
connect(4,[queue,2],[queue,3]).
connect(5,[queue,3],[server,1]).
connect(6,[server,1],[branch,1]).
connect(7,[branch,1],[departure,1]).
connect(8,[branch,1],[queue,4]).
connect(9,[queue,4],[server,2]).
connect(10,[server,2],[departure,2]).

start_time(0).
end_time(300000).
```

図３　ＳＩＬＱによるプログラム

ウインドウ・フロー制御は、パケット交換網において、各発信局間に設定される論理的な伝送路である論理チャンネル（ＬＣ）ごとに、網内に入力できる未送達パケット数を一定数（ウインドウ・サイズ）以下に規制し、網内の輻輳を緩和する制御方式である[14]。

図３に、このモデルをＳＩＬＱを用いて記述したプログラム例を示す。

３．ＳＩＬＱシミュレータの構成

３・１　開発方針

まず、シミュレータの開発言語についての検討を行う。第一の方針としては、ＳＩＬＱがPrologと同一の記述形式を持つことを考慮しシミュレータ全体をProlog を用いて開発する事が考えられる。この方法では、Prolog 処理系の上でユーザプログラムとシステムプログラムを結合することで実行可能なシミュレーションプログラムが生成でき、デバッグ、モニタリング等が容易であり、開発効率の高さが特徴と考えられる。また、ＳＩＬＱのシンタックスをシミュレータ内部でも利用し易いために、アルゴリズムは簡単になる。ここで、既存のProlog 処理系上でシステム構築を行うことを考えると、述語計算に必要となる処理履歴の保存が問題となる。ＳＩＬＱの実行をProlog で行えば、多くの観測サンプルを必要とするシミュレーションの性質上、シミュレーションの進行にともない、探索木は限りなく深くなり、処理履歴（使用メモリ量）は膨大になる。この問題を解決するためには、以後のシミュレーションの実行に必要な処理履歴だけを保存し、強制的なバックトラックによりメモリを解放する処理が頻繁に必要となる。本来、シミュレーションの実行においては保存の必要な処理履歴はほんの一部である。シミュレーションではこの様な処理のほとんどが無駄な処理となり、効率を大きく損なう結果となる。

そこで、汎用のコンパイラ言語を基本にしてシミュレータを構成し、ホーン節処理を行うProlog インタープリタの機能を内部に実装することによって、ＳＩＬＱ特有の記述の処理を行うことを考える。これについては、３・６で詳しく述べる。ＳＩＬＱシミュレータを開発する言語として、移植性と処理速度を考慮し、現在処理系の豊富なＣ言語[15][16]を用いる。

３・２　シミュレータの概要

本シミュレータは、事象駆動型のシミュレーション実行制御方式を用いる。待ち行列網モデルにおいては、構成要素（ＱＮプロセス）間の相互関連は比較的疎であり、生起する事象は時間経過に関して離散的に生じる。この性質を

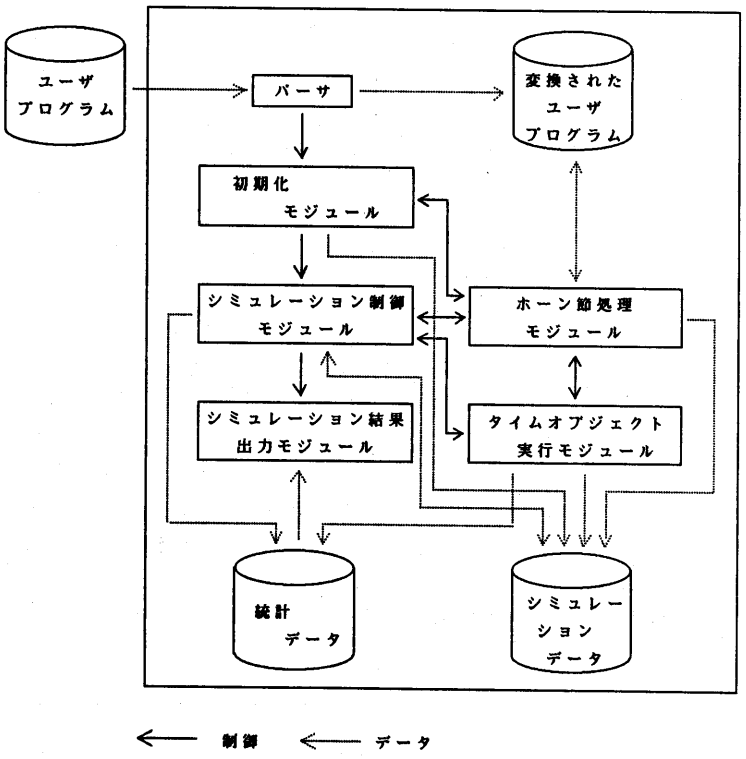

図4　シミュレータの基本構成

利用し生起事象の系列処理によってシミュレーションを実行する。シミュレータの基本構成は図4に示す通り、6つのモジュールと、内部表現に変換されたユーザプログラム、及び2つのデータ領域からなる。

3・3　処理アルゴリズムの概略

図5は、本シミュレータで採用したシミュレーション実行制御アルゴリズムの概略である。Arrival ルーチン等のＱＮプロセスルーチンは、いくつかの事象処理ルーチン群から構成されている。シミュレーションの実行は、シミュレーションデータ中から事象発生時刻の最も早いＱＮプロセスを逐次選び出し、対応する事象処理をＱＮプロセスルーチンで行うことで進められる。これについては、3・5で詳しく述べる。

3・4　データ構造

ＳＩＬＱシミュレータは、シミュレーションデータと統計データの2つのデータ領域を持つ。シミュレーションデータはシミュレーションモデルの構成要素の状況を表すデータであり、モデル内に存在する各ＱＮプロセスを表すデータと客を表すデータがある。統計データは各ＱＮプロセスそれぞれに対して生成され、そのプロセスに関する統計量算出に必要なデータを蓄積する。以下にシミュレーションデータの構造について述べる。

(1) Process Data

Process Data は、各ＱＮプロセス毎に生成され、次の情報を持つ。

1．ＱＮプロセスの識別子
2．通過した客が次に訪れるＱＮプロセス
3．次にこのＱＮプロセスが起動する時刻
4．同時に受け入れ可能な客数
5．統計データのポインタ

シミュレーション実行中、各ＱＮプロセスは以下に示す5つの状態のいずれかに属している。各 Process Data は、この状態によって分類され連結リストを構成している。

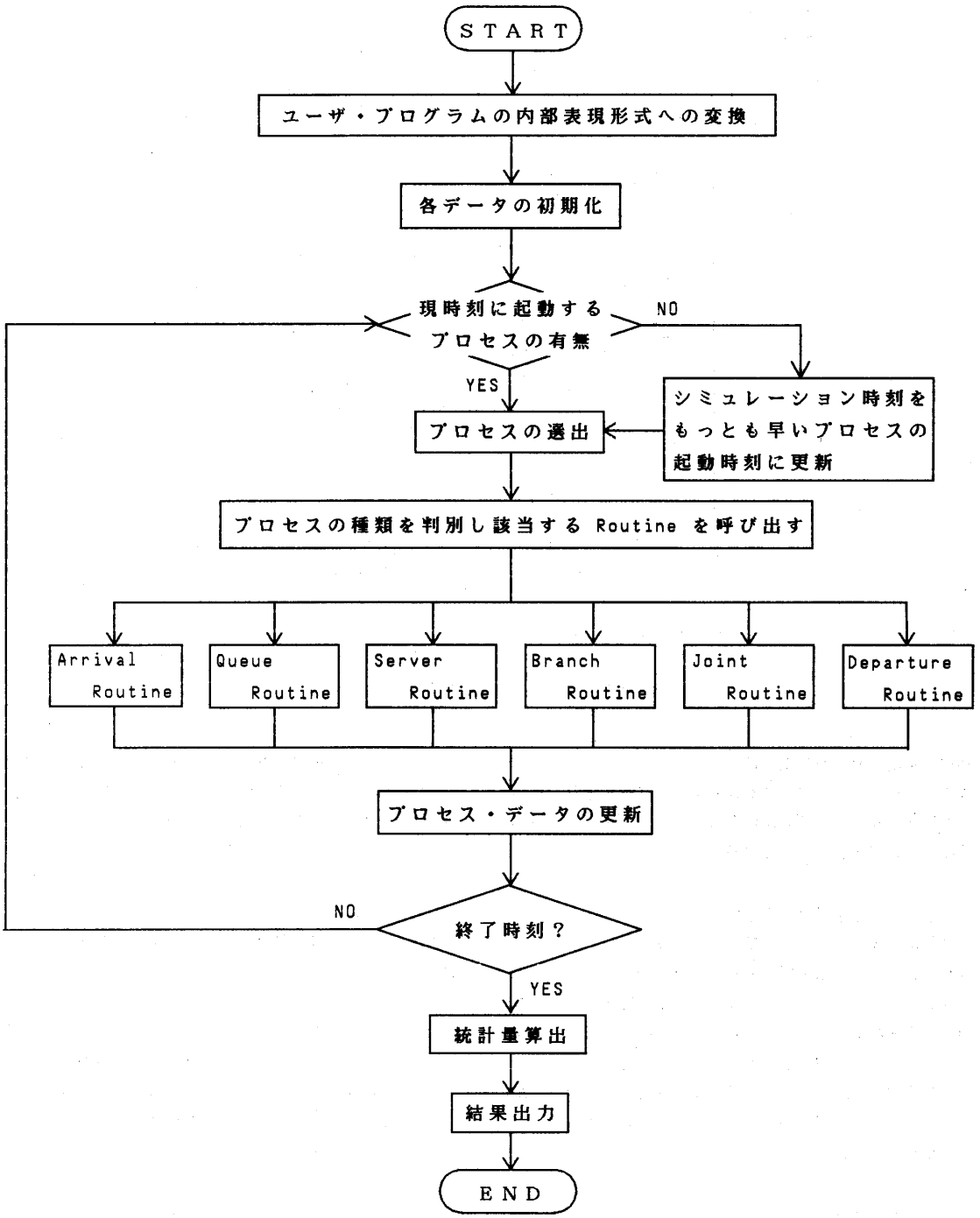

図5　ＳＩＬＱシミュレータの
処理アルゴリズムの概略

<Active State>
現在事象処理を実行中（常時１つ）
<Holding State>
決められた時間まで次の事象処理を待機中
<Message Waiting State>
客が到着するのを待機中
<Condition Waiting State>
次の事象処理のためある条件が満たされるのを待機中
<Time Waiting State>
客の到着または、時刻の更新直前まで事象処理を停止中

これらの内、Holding 状態にある Process Data は、次に起動する時刻順にソートされて格納されている。この様に、 Process Data がいまどこに格納されているかによってプロセスの状態が表現され、また、そのデータ集合によってシミュレーションの実行状態が表現されている。

(2) Customer Data

モデル内で処理を受ける客に関する情報である。ＳＩＬＱシミュレータにおいては、 １つの Customer Data は１人の客に対応している。 それぞれの Customer Data の内部には次の情報が蓄えられる。

１．一つ前に通過したＱＮプロセス
２．そのＱＮプロセスを出た時刻
３．発生した時刻
４．モデルフィールド
５．次に訪れるＱＮプロセス（移動中の場合）

この内モデルフィールドはユーザが客に関して自由に定義し書き込むことができる領域であり、プライオリティの設定・経路選択等の制御に用いることができる。

本シミュレータでは、シミュレーション実行中に客が発生消滅するたびにメモリの確保、解放を行うオーバヘッドを軽減するため、 初期化の段階で一定数のCustomer Data を生成しておき、書き換える事で再利用する。 従って、Customer Dataは以下に示す３つの状態のいずれかに属する。

Ⅰ 利用されず休止中の状態
Ⅱ 次のＱＮプロセスに移動中の状態
Ⅲ あるＱＮプロセスの中に滞在中の状態

Ⅰの状態にある Customer Data は一つの連結リストとして格納される。Ⅱの状態にある Customer Data は、その移動先プロセスの種類ごとに連結リストで格納される。また、Ⅲの状態にある Customer Data は、 各ＱＮプロセスごとに一つの連結リストを構成する。

３・５ 離散シミュレーションにおけるプロセス処理

本シミュレータのＱＮプロセス処理は、シミュレーション制御モジュールで事象が発生するＱＮプロセスを逐次選出し、タイムオブジェクト実行モジュール内のプロセスルーチンで対応する事象処理を行うことによって行われる。

タイムオブジェクト実行モジュールは、６つのプロセスルーチンからなり、各々のプロセスルーチンは幾つかの事象処理ルーチンを持つ。

以下ではＱＮプロセスの選出方法とプロセスルーチンにおける事象処理について更に詳しく述べる。

（Ｉ）ＱＮプロセスの選出

シミュレータが現在のＱＮプロセスの処理の後、次にどのＱＮプロセスの事象処理を行うべきかは、Process Data、Customer Data の状態をもとに、以下に示す優先順位に従って、 シミュレーション実行モジュールにおいて決定される。この選出の過程において、「待ち行列から先頭の客を出す」という事象が発生する条件といった、ユーザプログラムに記述されている情報が必要になればホーン節処理モジュールが起動される。

プロセス選出の優先順序

1 Holding State にあり、かつ現在の論理時刻に次の事象が生起するＱＮプロセス
2 Message Waiting State にあるＱＮプロセスで、そのＱＮプロセスに向かって移動中の客があるもの
3 Time Waiting State にあるＱＮプロセスで、そのＱＮプロセスに向かって移動中の客があるもの
4 Condition Waiting State にあるＱＮプロセスで、ユーザプログラムで指定された次の事象発生条件が満たされたＱＮプロセス
5 Time Waiting State にあるＱＮプロセス（時刻更新までに客の到着がなかったものと判断する）
6 Holding State にあるＱＮプロセスの中で、事象の発生する時刻のもっとも早いプロセス（その時刻に論理時刻を進める）

（Ⅱ）プロセスルーチンにおける事象処理

事象処理を行うべきＱＮプロセスが選ばれると、タイムオブジェクト実行モジュール内の、該当するＱＮプロセスルーチンが起動される。ＱＮプロセスルーチンでは、選び出されたＱＮプロセスの Process Data から次の事象を判断し、該当する事象処理ルーチンを起動する。

事象ルーチン内では、生起事象に応じて、関連するシミ

ュレーションデータ及び統計データの書換えが行われる。このときに、サービス時間、待ち行列の優先権等に関するユーザプログラム中の記述が処理の判断に用いられる。この判断は、ホーン節処理モジュールを介して所望の情報を得た後行われる。事象処理の終ったＱＮプロセスは停止し、Process Data は適当な位置に格納されて、次の事象生起を待つ。

３・６　ホーン節処理

ユーザプログラムは、初期化の段階で内部表現形式に変換される。ホーン節処理モジュールはＰrologインタプリタと同様の機能を持ち、シミュレーション実行中、ユーザプログラムに依存した処理を行う必要がある場合には、シミュレーション実行モジュール、もしくはタイムオブジェクト実行モジュールからの質問をホーン節処理モジュールが処理する[17]（図６）。

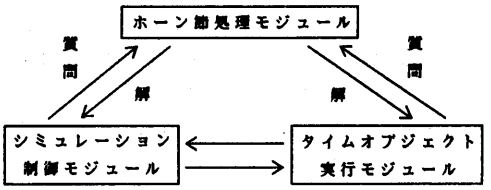

図６　ホーン節処理モジュールの起動

（Ｉ）ユーザプログラムの内部表現

ユーザプログラム内の各々の節は、図７に示すように、ホーン節の本体部にダミーゴール True が付け加えられ、さらに２引数の項の形に変換される。

H：－G1, G2.
↓
H：－G1, G2, true.
↓
':-'(H, ','(G1, ','(G2, true)))

図７　節の変換

項の形に変換された節は各々以下に示す３つの領域によって内部表現される。

（１）記号領域（Symbols）

節に現れる記号（関数記号、述語記号、定数、':-'、等）及び、その記号に関する情報（引き数の数等）

（２）節領域（Cells）

項の形に変換した節を表現する。

（３）節情報領域（Clause-header）

各節の節領域での位置に関する情報

図８に、項の形に変換した節の内部表現の例を示す。図８において、節に現れる大文字は変数を示す。記号領域には、項を表現するのに必要な記号,'interval','arrival1' 'exp_dis'、'10'が書き込まれている。なお、記号領域の１～４に示されている記号はシミュレータが予め用意している。節領域では、項の引き数２個と、記号１個の計３つのセル（１０１～１０３）によって項が表現されている。第１０１セルは記号':-'の記号領域でのアドレスが１であることを示している。項の引き数も同じく項の形をしており、同様に節領域に表現されている。第１０２セル、第１０３セルは各々、第一引き数、及び第二引き数の項が表現されている節領域の先頭のアドレスを示している。節情報領域には、この節が節領域でどの位置に書き込まれているかが示されている。この例では、節領域情報のnext.adはnilになっている。これは、intervalを定義する節が他に存在しないことを示している。もしも、他にintervalを定義する節が存在する場合には、その節情報領域のアドレスがnext.adに書き込まれる。

':-' (interval(arrival1,T), (exp_dis(10,T),true))

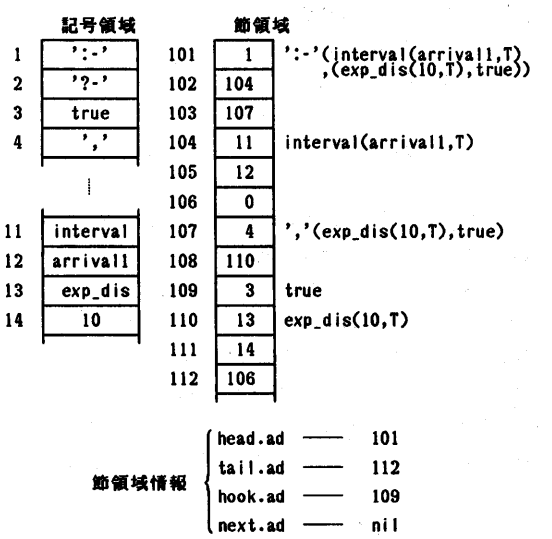

図８　節の内部表現例

（Ⅱ）ホーン節処理モジュール

ホーン節処理モジュールは、ホーン節形式の質問を文字列として与えられることによって起動する。このモジュールは、与えられた質問をゴールとして、ユーザプログラムの内部表現をもとに、structure-copying方式[18]によって述語計算を行い、質問に対する解を返す。

4．シミュレータの評価・検討

4・1　ユーザインターフェースの検討

本ＳＩＬＱシミュレータによる実行結果の出力例を図９に示す。対象とするモデルは、図２のウインドウ・フロー制御を実装した２サーバ直列網モデルである。

現在のところ本シミュレータは、シミュレーション実行中の途中経過に関する表示機能を持たない。しかし、実用化を考えれば、シミュレーションの実行状況を視覚的に表示する等の表示機能の充実によって、モデル記述の妥当性に対する検証支援、モデルの過渡的な振舞いの観察等を容易にすることができると考えられる。このため、種々の表示フォーマットの中からユーザが任意に選択するなど、途中結果を表示する方式について今後検討が必要である。

```
***  SILQ simulator summary report  ***

simulation --> window.pro
current time 100002

ARRIVAL 1
  arrived customer count   =       3318
  average arrival interval =     30.145
ARRIVAL 2
  arrived customer count   =       4964
  average arrival interval =     20.146
QUEUE 1
  passed customer count    =       3219
  average queueing time    =      9.128
  average queue length     =      0.294
  maximum queue length     =          3
QUEUE 2
  passed customer count    =       4512
  average queueing time    =     14.244
  average queue length     =      0.643
  maximum queue length     =          3
QUEUE 3
  passed customer count    =       7730
  average queueing time    =     19.273
  average queue length     =      1.490
  maximum queue length     =          4
QUEUE 4
  passed customer count    =       4512
  average queueing time    =      5.580
  average queue length     =      0.252
  maximum queue length     =          2
SERVER 1
  passed customer count    =       7730
  average service time     =     10.550
  utilization              =      0.816
SERVER 2
  passed customer count    =       4512
  average service time     =     10.536
  utilization              =      0.475
DEPARTURE 1
  departed customer count  =       3217
  average delay time       =     38.869
DEPARTURE 2
  departed customer count  =       4512
  average delay time       =     60.245
```

図９　ウインドウ・フロー制御モデルの実行結果

4・2　シミュレータの妥当性の評価

本シミュレータの妥当性に関して実行結果の精度を用いて評価を行う。ここで、数学解析によって厳密解を得ることができるM／M／１待ち行列モデルを用いて、理論値とシミュレーション値との比較を行う。平均待ち行列長についての比較結果を図１０に示す。なお、シミュレーションにおけるサンプル数は約１万個である。ここで、理論値との若干の差異はシミュレータの持つ乱数発生のばらつきによって生じると考えられるが、本シミュレータの結果は十分信頼できるものであり、本シュミレータは正当であると結論できる。

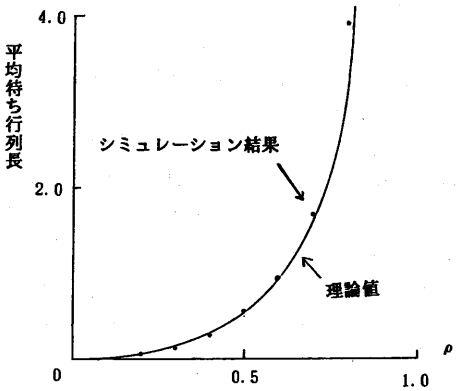

図１０　理論値との比較

4・3　処理効率の評価

シミュレータの処理効率を検討するために、Ｐrologで開発したプロトタイプとの比較を行った。パーソナルコンピュータＰＣ－９８０１VM上において、同一モデルに対するシミュレーションを行い、両シミュレータの処理時間、及び速度の比較を行った結果を表２に示す。使用したモデルは、図２のウインドウ・フロー制御を行った場合の２サーバ直列モデルであり、サンプル数は約１千個である。

表２　処理速度の比較

	退去客数	所要時間	所要時間／客
本シミュレータ	(LC1)　４８４	９分２１秒	０．４８秒
	(LC2)　６８４		
プロトタイプ	(LC1)　４７２	１時間５２分２２秒	５．７７秒
	(LC2)　６９７		

この比較結果より、Ｐrolog で開発したプロトタイプに比べ約１２倍の処理速度を実現していることが判明した。また、パーソナルコンピュータ上でのシミュレーション実行時間が客当り約０．５秒であることから、 大型計算機等への移植を考慮すれば、本シミュレータは実用システムとしての利用も可能と考えられる。

なお、現状では、本シミュレータは、モデルの動的な変化に対する対応が不十分である。例えば、ＳＩＬＱによるモデル記述では、論理時刻によってサービス窓口の大きさ

を動的に変化させることが可能であるが、本シミュレータではこれを実行することができない。これは次のような理由からである。この機能を実現するためには、現状では事象処理毎にホーン節処理モジュールを起動し、サービス窓口の大きさを求めることが必要となる。この様に、動的なモデルに対応しようとするとホーン節処理モジュールへの負荷が大きくなり、シミュレータの実行処理速度を著しく低下させる。

本シミュレータのより一層の高速化を図るには、ホーン節処理モジュールの改善が必要である。例えば、本シミュレータのホーン節処理モジュールは、他のモジュールが質問を文字列として与えることによって起動する。これは、一般的なProlog インタプリタと同じ形態であるが、シミュレーション実行中に各モジュールから発せられる質問の種類及び内容は限定されており、かつユーザの与えるモデル記述自体は実行中に変化はしない。従って、動的なモデルに対する対応をも含めて、実行レベルでなんらかの最適化を初期化の段階で行っておくことによりオーバヘッドの軽減が可能と考えられる。

５．まとめ

Prologを記述形式として持つ待ち行列網シミュレーション言語ＳＩＬＱのための処理系の開発を報告し、システムが比較的簡単なモデルに対しては実用的速度で動作することを示した。

本ＳＩＬＱシミュレータは、Prolog形式で記述されたユーザプログラムを扱うためのホーン節処理機能を内部に持ち、逐次処理系との結合により効率化を図っている点が特徴である。この結果、述語計算に必要な処理履歴の保存及び消去を必要最小限に抑えることができ、Prologで開発したプロトタイプに対し約１２倍の処理速度の向上を実現した。また本シミュレータは、Ｃ言語によってソースプログラム（約２８００行）を開発しており、処理系が豊富で移植性が高い。本シミュレータの開発は、パーソナルコンピュータ及びＵＮＩＸワークステーションＡＳ－３０００上で行われ、現在、大型計算機への移植を進めている。

本シミュレータの妥当性については、本稿ではM／M／１待ち行列モデルの理論値との比較によって検証を行った。処理速度については、現状でも実用に耐えうると考えられるが、今後ホーン節処理モジュールを改良することでより一層の改善が可能であると考えている。また、動的なモデルに対する処理がまだ不完全であるが、これには、実行速度を低下させることのない処理の実現法の開発が必要である。更に、出力結果の表示方法を含めた実行状態の表示機能の向上等のユーザインターフェースに対する検討及び実現も今後の課題である。

謝辞

本研究において熱心な御討論を頂いた大阪大学大学院生小林 真也氏、荒木　大氏に感謝いたします。

参考文献

(1) Leonard Kleinrock ：“QUEUE SYSTEM､Volue 1､2：Theory”

(2) Emshoff,Sisson著 関根訳 ：“シミュレーション”、日本コンピュータ 協会 (1979)

(3) Naylor,Balintfy,Burdick 水野、小柳訳 ：“コンピュータシミュレーション”、 培風館 (1971)

(4) Stephen S.Lavenberg ： “Computer Performance Modering Handbook”

(5) 渡辺 ：“待ち行列網シミュレーション用論理型言語とその処理系に関する研究”、大阪大学学位論文 (1986)

(6) 荒木、渡辺、中西、真田、手塚 ：“論理型言語による待ち行列網シミュレ ーションについて”、信学技法 IN86-49 (1986)

(7) 荒木 ：“論理型待ち行列網シミュレーション言語とその処理システムに関する研究”、大阪大学卒業論文 (1986)

(8) 中島 ：“Prolog”、 産業図書 (1983)

(9) 中西 俊男 ：“シミュレーション言語”,情報処理、Vol.22, No.6(1981)

(10) 森戸、相沢 ：”ＳＬＡＭⅡによるシステム・シミュレーション入門”、構造計画研究所 (1986)

(11) 山本、浦 ：“離散型シミュレーション言語の現状と将来の展望(1)～(2)”、情処学会誌 vol22 , No.9 , No.11(1981)

(12) Sauer,MacNair ： “A Language for Extended Queueing Network Models”,IBM J. RES. DEVELOP Vol.24 No.6 (1980)

(13) Futo,I. Seredi,J. ： “A Discrete Simulation based on Artificial Interigence method” , Discrete simulation and related fields IMACS (1982)

(14) Kleinrock, Gerla ：“Flow Cotrol:A Comparative Survey” IEEE Tran. Commun. Vol.COM-28 No.4 (1980)

(15) Kernighan,Ritchie 石田著 ：“プログラミング言語Ｃ”、共立出版(1981)

(16) Herbert Schildt ：“ADVANCED C”、McGraw-Hill (1986)

(17) 玉木 ：“Prplog処理系入門 Pascalによるミニインタープリタ”、bit, vol. 18, no. 10

(18) J.A.Campbell ：“implementation of PROLOG” Ellis Horwood(1984)