ペトリネットによる交換SDL仕様の検証方式[†]

長谷川 晴 朗*

A Method of Verifying a Switching SDL Specification by Utilizing Petri Nets

Haruo HASEGAWA*

This paper describes a method of verifying an SDL specification of a switching system by utilizing the algebraic properties of Petri Nets(PN). In order to verify the SDL specification, it is converted into PN, which have been an excellent tool for modeling an asynchronous and parallel system. SDL is based on a finite state machine(FSM) model and FSM can be easily converted into PN. However, SDL has other functions than FSM such as creation and deletion of a process instance, or realization of a synchronous communication between processes by adding a 'wait' state. The SDL specification is converted into PN considering these functions. In this paper, Colored Petri Nets(CPN) are used because they have more powerful modeling ability than conventional Petri Nets. Two kinds of invariants, S-invariant and T-invariant, can be calculated from the incidence matrix of the created Petri Nets which means connections between transitions and places. Then, boundness of the specification is verified by S-invariants, and liveness and deadlock-freeness by T-invariants. It is very difficult to get the invariants from the incidence matrix in CPN, because the elements of the matrix are not integers. However, the proposed method can utilize the invariants got from the incidence matrix of conventional PN, which are created from CPN. Besides, in a case that engineers design the SDL specification incrementally, they can ensure the correctness which the specification had before they add new services to it, by checking the invariants. Finally, the method is successfully applied into design of a PBX prototyping system.

Key Words : Petri Nets, SDL, specification verification, invariant

1. はじめに

近年の通信システムの高度化・多様化・高速化に伴い、それを構築するソフトウェアがますます大規模になってきている。新しいソフトウェア開発のパラダイムでは、仕様さえ設計・検証・保守すればその後のソースプログラムへは自動的に変換できるとしており¹⁾、仕様化工程における検証が一層重要なものとなっている。このパラダイムに従って、ITU-TS(国際電気通信連合-電気通信標準化セクター、なお1993年3月まではCCITT)より通信システム用仕様記述言語として勧告されているSDL²⁾をベースとした、通信ソフトウェアの開発支援シ

ステムが各機関で開発されている^{3)~5)}。

SDL仕様を検証するものとしては、SDLの文法に適用かどうかを調べるSDLチェッカ、SDLのままで実行してシグナルの流れや変数の値が設計者の思うとおりになっているかを調べるシミュレータなどがある。さらに、いったんSDL以外の言語による記述に変換して、その段階で種々の性質を調べるものが各機関で考えられている。その中では、従来より非同期・並行系システムのモデル化・検証用ツールとしてのペトリネット^{6),7)}を利用したものが挙げられる^{8)~10)}。しかし、それらは、到達木を作成して到達可能性を調べたものが多く、数学的な性質を用いて論理的に検証したものはきわめて少ない。本稿では、SDL仕様を通常のものよりモデル化能力に優れる色付きペトリネット¹¹⁾に変換し、ペトリネットの構造的な性質を示す接続行列から求めたSインバリエントとTインバリエントを利用して、各種の解析を行う¹²⁾。

まず、Sインバリエントからは、全体のペトリネットが

[†] 第9回離散事象システム研究会(T2)で一部発表(1992・7)

* 沖電気工業(株) 千葉市美浜区中瀬2-6

* Oki Electric Industry Co., Ltd., Chiba
(Received August 3, 1992)
(Revised August 18, 1993)

S インバリエントによって被覆されていることによって有界性が成立することを示す。つぎに、T インバリエントからは、全体のペトリネットが T インバリエントで被覆されていることによって活性が成立することを示す。さらに、仕様はただ 1 回の設計で完成するものではなく、インクリメンタルに設計していくものであり、その場合仕様を追加する前のものが仕様追加によりその正当性を失っていないかを、それらのインバリエントが新しいペトリネットでも求められることより調べる。

以下では、本論文で扱う SDL モデル、ペトリネットの基本用語とそれを使用した解析手法、SDL 仕様の検証項目とペトリネットでの検証方法、SDL からペトリネットへの変換方法について記した後、その検証法を PBX の簡単な例に適用した結果について報告する。

2. 通信システム用仕様記述言語 SDL

2.1 SDL とは

SDL は、通信システムの要求動作(required behaviour)の仕様化(specification)と、実際の動作(actual behaviour)の記述(description)のための言語として ITU-TS より勧告されており、全世界の各機関で広く使用されている。なお、SDL については付録で簡単に触れるが、詳細説明は文献 2) を参照されたい。

ここでは紙幅の都合から、本論文で述べようとする点について誤解を招かない程度に SDL をかなり簡略化して記述しており、必ずしも SDL の正しい文法に従ってはいないことを断っておく。

2.2 PBX モデル

SDL 仕様の検証については、対象とする PBX (Private Branch eXchange) がどのような SDL モデルであるかを記述する必要がある。ここでの SDL モデルは、端末からの操作によって交換システムがどのように状態遷移するかを表現するものである。したがって、端末や PBX 内部のハードウェアおよびソフトウェアといったリソースごとにプロセスを設け、それらのプロセス間のシグナルの送受により各プロセスが状態遷移していく方式としている。

ここでは、Fig. 1 に示すように PBX をモデル化し、これに従って SDL 仕様が作成されている。このモデルでは、PBX を 3 ブロックに分ける。一つは、端末やトランクのように、PBX の外部に接続される装置を論理的に表現したものと、それらをサービスとして管理するものからなる call-processing (以下、cp と略す) ブロックである。第 2 は、呼出信号や会議用トランクのような PBX 内のハードウェアリソースおよびそれらを管理するものからなる resource (以下、rs と略す) ブロックである。最後

Fig. 1 SDL model of PBX

のものは外部ハードウェア装置を監視・制御する device (以下、dv と略す) ブロックである。dv ブロックは、ハードウェア・アーキテクチャに大きく依存するものであり、cp ブロックとの間でそれに応じた監視・制御信号がシグナルとして送受される。dv ブロックは、後述のペトリネット変換への関連性が少ないため詳細の説明は割愛する。

cp ブロックは、内線加入者の電話機端末を意味する ext プロセス、局線や専用線との接続インタフェースであるトランクを示す trk プロセス、およびそれらを管理する terminal-management (以下、tm と略す) プロセスからなる。rs ブロックも同様に、呼出信号を意味する rg プロセス、会議トランクを意味する cft プロセス、およびそれらを管理する resource-management (以下、rm と略す) プロセスからなる。ext, trk, rg, cft の各プロセスをリソースプロセスと呼び、tm, rm のプロセスを管理プロセスと呼ぶ。システムの生成時には、管理プロセスのみが存在し、リソースプロセスのインスタンスはまだ存在していない。呼が発生しリソースが必要となったときに対応するプロセスインスタンスが生成され、呼が終了するなど不要となったときに消滅する。リソースプロセスのインスタンスが生成された後は、他ブロックとのシグナルの送受は管理プロセスを介さずに直接行われる。管理プロセスは、これらのインスタンスの生成・消滅管理のほかに、ダイヤル信号の番号翻訳などの処理を行う。

打合せ通話(Consultation Talk)と 3 者の会議通話(Three-way Conversation)を含む内線相互接続サービス(Station-to-Station Dialing Service)に関する SDL 仕様を考える。まず、最も単純な 2 者間の内線相互接続の仕様について記す。内線加入者(ext: extension)が最

初のアイドル状態からオフフック(offhook)すると、ダイヤル音(dt: dial tone)が聞こえる。その状態で相手の電話番号をダイヤルすると呼出音(rbt: ringback tone)が聞こえ、相手には呼出信号(rg: ringing signal)が送出される。相手がオフフックして応答すると2者間の通話に入る。そして、片方がオンフック(onhook)するととのアイドル状態に戻り、相手には話中音(bt: busy tone)が聞こえる。その状態でオンフックするととのアイドル状態に戻る。

打合せ通話とは、2者間の通話からフッキング(hooking)を行うことにより、サービスダイヤル音(sdt: service dial tone)を聴取してほかの内線加入者をダイヤルして呼び出し、応答後打合せをして、その相手のオンフック操作でもとの通話に戻るサービスである。なお、打合わせ通話中、それまでの通話相手に保留音(ht: hold

tone)を聴取させている。また、3者会議通話とは、打合せ状態からフッキングを行って3者間の通話に入るものである。そのためには3者以上による同時会話が可能な会議トランク(cft: conference trunk)が必要である。

ここでは、SDL仕様として、上記の内線相互接続サービスの仕様全体の流れを比較的わかりやすく表現しているextプロセス図の中で、2者間の通話状態に入ってからのものみをFig. 2に示す。これは、2者間の内線相互接続は様々な論文で触れられているが、現実に重要なのは打合せ通話や会議通話のような具体的なサービスだからである。なお、Fig. 2で各行の右端に“conference”および“conf. end”と記した部分が、それぞれ3者会議接続、3者会議終了に関する仕様である。

各プロセスインスタンスでは、接続されている相手側(ext, rg など)のPidを知っており、それを指定してシグナルを送信する。たとえば、Fig. 2のmateext1Pidやmateext2Pidは、その内線加入者に接続されている相手側のextプロセスインスタンスのPidを示す。これらのPid値はインスタンス生成時に送受されるが、本論文の内容と直接には関係なくまた同図が複雑になるのを避けるため、そのためのシグナルの送受を記していない。

3. ペトリネット

本論文では、端末の操作を起動イベントとした交換システムの状態遷移に着目し、端末を通してシステムを利用するユーザから見たサービス仕様の観点で仕様をとらえる。それを記述するペトリネットに関して、通常のものおよび色付きペトリネットの順序に以下に記す。

3.1 通常のペトリネット

ペトリネットは、非同期・並行系システムのモデル化および解析のためのツールであり、現在各方面で使用されている。構成はきわめて簡単で、事象生起に関するシステムの条件を示すプレース、離散事象の生起と完了を示すトランジション、およびそれらの間を結ぶアークからなる。そして、プレースに対応する条件が現時点で成立している場合には、そのプレース中にトークンを置き、トランジションの発火に基づくトークンの移動によってシステムの状態遷移を表わす。各プレースにトークンが設置された状態をマーキングという。主要な用語の定義を以下に記す。

(a)構成要素 ペトリネットを4項組、 $N=(P, T, A, M)$ で表わす。ここで、 P, T, A, M はそれぞれ、プレース(p)の集合、トランジション(t)の集合、アークの多重度を示す接続行列(行: トランジション, 列: プレース, $A=A^+-A^-$; A^+ は T から P への行列, A^- は P から T への行列), マーキング(列ベクトル)である。

```
PROCESS ext (0);
.....
STATE idle; /* idle state */
INPUT con_rgl; /* from mateext1Pid */
NEXTSTATE ext_rgl;
STATE ext_rgl; /* ringing state for consultation talk */
INPUT offhook; /* from dv */
OUTPUT con_ext_t TO mateext1Pid;
NEXTSTATE ext_ext2;
STATE ext_ext; /* two-way conversation */
INPUT hooking; /* from dv */
OUTPUT con_ht TO mateext1Pid;
NEXTSTATE ext_sdt;
INPUT con_ht; /* from mateext1Pid */
NEXTSTATE ext_ht;
STATE ext_sdt; /* flash the switchhook and dial the third party */
INPUT hooking; /* from dv */
OUTPUT recon_ext TO mateext1Pid;
NEXTSTATE ext_ext;
INPUT dial; /* from dv */
OUTPUT req_anal TO tmPid;
NEXTSTATE wait1;
STATE wait1; /* analysis of the dialed number */
INPUT con_rbt; /* from tm */
OUTPUT con_rgl TO mateext2Pid;
NEXTSTATE ext_rbt1;
STATE ext_rbt1; /* ring the third party */
INPUT con_ext_t; /* from mateext2Pid */
NEXTSTATE ext_ext1;
STATE ext_ht; /* connection of a hold tone */
INPUT recon_ext; /* from mateext1Pid */
NEXTSTATE ext_ext;
INPUT con_cft; /* from mateext1Pid */ /* conference */
NEXTSTATE ext_cft; /* conference */
STATE ext_ext2; /* consultation talk (the calling party) */
INPUT onhook; /* from dv */
OUTPUT recon_ext TO mateext1Pid;
OUTPUT recon_ext TO mateext2Pid;
STOP;
INPUT con_cft; /* from mateext1Pid */ /* conference */
NEXTSTATE ext_cft; /* conference */
STATE ext_ext1; /* consultation talk (the called party) */
INPUT recon_ext; /* from mateext2Pid */
NEXTSTATE ext_ext;
INPUT hooking; /* from dv */ /* conference */
OUTPUT hnt_cft TO tmPid; /* conference */
NEXTSTATE wait2; /* conference */
STATE wait2; /* hunt of conf. trunk */ /* conference */
INPUT con_cft0; /* from tm */ /* conference */
OUTPUT con_cft TO mateext1Pid; /* conference */
OUTPUT con_cft TO mateext2Pid; /* conference */
NEXTSTATE ext_cft; /* conference */
STATE ext_cft; /* connection of conf. trunk */ /* conf. end */
INPUT onhook; /* from dv */ /* conf. end */
OUTPUT rm_cft TO mateext1Pid; /* conf. end */
OUTPUT rm_cft TO mateext2Pid; /* conf. end */
OUTPUT rm_cft0 TO cftPid; /* conf. end */
STOP; /* conf. end */
INPUT rm_cft; /* from mateext1Pid */ /* conf. end */
NEXTSTATE ext_ext; /* conf. end */
.....
```

Fig. 2 A part of 'ext' process in SDL specification of a station-to-station dialing service

(b) 発火条件・発火系列 あるマーキング M_0 で j 番目のトランジション T_j が発火するための条件は, T_j に入力する各プレースの中のトークンが, T_j とそれらのプレースとを結ぶアークの多重度以上に存在することである. したがって, j 番目のみが 1 でその他の要素がすべて 0 である単位列ベクトルを e_j , A^{-T} の転置行列を A^{-T} とおいて, 次式が成立することが必要十分条件となる.

$$M_0 \geq A^{-T} \cdot e_j \quad (1)$$

また, M_0 から別のマーキング M に至る発火系列で, 各トランジションの発火回数を示す発火回数ベクトルを γ とすると, トランジションの発火ごとに, 各入力プレースからアークの多重度に等しい数のトークンが抜け, 各出力プレースにアークの多重度に等しい数のトークンが入ることから, (2) 式が成立する.

$$M = M_0 + A^T \cdot \gamma \quad (2)$$

(c) S インバリエント ある列ベクトル y が次式を満足するとき, これを S インバリエントと呼ぶ.

$$A \cdot y = 0 \quad (3)$$

(2) 式の両辺に y^T を左から乗じると, $y^T \cdot A^T = (A \cdot y)^T = 0$ であるから, M_0 から到達可能なすべてのマーキング M に対して, $y^T \cdot M = y^T \cdot M_0 = \text{一定}$ となる. つまり, プレースごとにある重み付けをしたトークンの総和が, マーキング状態によらず一定であることを示している. なお, 本論文では各要素は非負整数とする.

(d) T インバリエント ある列ベクトル x (各要素は非負整数) が次式を満足するとき, これを T インバリエントと呼ぶ.

$$A^T \cdot x = 0 \quad (4)$$

T インバリエントは, あるマーキングから再びそのマーキングに戻るまでの発火回数ベクトルを表わす. 特に, T インバリエントのうちで和分解不能なものを初等的 T インバリエントという. なお, トランジションの発火には (1) 式の成立が必要なため, T インバリエントの存在は発火系列存在のための必要条件であって十分条件ではない. 逆に, トークンが十分存在すると仮定したとき, それらは必要十分条件の関係にあるといえる.

S および T インバリエントは (3) 式や (4) 式の多元 1 次方程式を解くことにより得られるが, それらを高速に求めるアルゴリズムが研究されている¹³⁾.

(e) サービス あるマーキングから再びそのマーキングに戻るまでのトランジションの発火系列をサービスと定義する. つまり, T インバリエントは, 一つのサービスにおける各トランジションの発火回数を意味する. 特に, 初等的 T インバリエントに相当するサービスを素サービスと定義する. 素サービスの中で, 一巡してもとに戻るマーキングが初期マーキング(各リソースがアイ

ドル状態にあることを示すマーキング)であるとき, そのサービスを完結サービスと呼ぶ. 反対に, 完結サービスでない素サービスを非完結サービスと呼ぶ. すべてのサービスは完結サービスと非完結サービスの線形結合, 特に和による線形結合で表現できるため, 素サービスをサービスの基本単位ととらえることができる.

3.2 色付きペトリネット

色付きペトリネットとは, 通常のペトリネットを拡張したものであるハイレベルネットの一つであり, プレース, トランジションやアークなどに識別子(色)をもたせて発火やトークンの出入りを制御するものである. ここでは以下の理由から通常のペトリネットではなく色付きペトリネットを考える.

- (a) 正確かつ簡明に仕様を記述できる.
- (b) 様々なリソースを明確にモデル化できる.
- (c) リソース間の関係を明示的に記述できる.

色付きペトリネットにおいても, 3.1 節で述べた種々の性質が通常のペトリネットと同様に成立する. ただ, 色付きペトリネットでは, インバリエントを求めるアルゴリズムに関する研究がなされているが¹⁴⁾, 必ずしも確立されていない. ただ, 後述するようにここで取り扱うものについては, 通常のペトリネットとして求めたインバリエントをほぼそのままの形で利用することが可能である.

4. 仕様検証

4.1 仕様の検証項目

通信システム仕様の検証項目としては, 以下のものがあげられる.

- (a) 完全性 あらゆる可能な入力処理され, かつあらゆる出力が受信されること.
- (b) 有界性 メッセージの数およびバッファが有限であること. 特にその数が 1 のとき安全性という.
- (c) 活性 初期状態から始まって, 状態遷移が実行可能であること.
- (d) デッドロックのないこと ほかのいかなる状態へも遷移できない状態とならないこと.
- (e) 初期状態への復帰 交換システムの特徴としてあらゆる状態から初期状態に復帰できること.
- (f) 無矛盾性 仕様の中に操作が不可能というような矛盾がないこと.
- (g) 要求充足性 ユーザ要求が仕様に正しく反映されていること.

4.2 ペトリネットでの検証方式

上記項目のうち, (f) および (g) については仕様の意味的な側面に立ち入ることが必要であり, 本稿で対象と

する論理的な検証の範囲には入らない。また、(a)についてはペトリネットへの変換の過程で自動的にチェックされるため、変換後の検証の対象は(b), (c), (d)および(e)である。以下に、ペトリネットの代数的な性質を利用した検証方式^{15),16)}について述べる。4.1節で列挙した検証項目の中で、Sインバリエントを用いて(b)を、Tインバリエントを用いて(c), (d)および(e)を検証する。

4.2.1 Sインバリエント

3章で述べたように、Sインバリエントはトークンの保存性について規定しており、Sインバリエントで0でない要素に対応するプレースは有界である。さらに、ペトリネット全体がSインバリエントで被覆されていれば、すべてのプレースが有界であるといえる。

4.2.2 Tインバリエント

Tインバリエントは、ある状態から再びその状態に戻るまでのサービスにおけるトランジションの発火回数を示す。したがって、完結サービスを意味するTインバリエントで、0でない要素に対応するトランジションは活性である。また、完結サービスの生成するマーキングの途中から非完結サービスが始まるとき、非完結サービスを意味するTインバリエントの0でない要素に対応するトランジションも活性である。したがって、ペトリネット全体がTインバリエントで被覆されていれば、すべてのトランジションが活性であるといえる。

4.2.3 インクリメンタルな設計

われわれがソフトウェアを設計するとき、ゼロの状態から一気に全体を作成することはきわめて少ないといってよく、現実には少しずつ作成してはその仕様を検証し、その後再び設計を繰り返すことが多い。つまり、このようなインクリメンタルな設計を行う場合、追加設計した部分に誤りが含まれ得ることは充分考えられることとして、本来追加した箇所に関係がないはずの部分にまで誤りを生成してしまうことに注意を要する。そのような場合、前述のインバリエントに関する諸性質を利用して追加する前のインバリエントを求め得るかどうかにより、追加の前に有していた部分が不変であることを検証することができる。

5. SDL からペトリネットへの変換方式

5.1 SDL仕様とペトリネット仕様

SDL仕様は設計者の立場から見た設計仕様であり、ソフトウェアを作成していく上で最も基本的なドキュメントとなるものである。一方、ペトリネット仕様はシステムの中の動作について何らの関心も知識もないユーザの立場に立って、あくまでも外部からの端末などの操作と

Fig. 3 Conversion example of SDL with parallelism

それに対するシステムの応答との組合せとしてみたサービス仕様であるといえる。

SDLでは、1種類のプロセスは複数のインスタンスをもち得るが、ペトリネットではインスタンスの概念がなく色付きペトリネットでそれを表現する。また、SDL仕様では複数個のプロセスが並列に動作可能であることを利用して、各リソース単位にプロセスが存在し、それらが通信を行いながら並列に動作している。ここで状態遷移にのみ着目すると、それをペトリネットでは自然な形で表現することができる。簡単な例をFig. 3に示す。これは同一の状態遷移をSDL、ペトリネットで表わしたものである。SDLではプロセス1とプロセス2との間でシグナルを送受信し、かつプロセス1ではプロセス2からのシグナルを受信するための状態として“wait”を設ける必要があるのに対して、ペトリネットでは実にシンプルに同期を表現することができる。

5.2 基本的な変換ルール

SDLからの変換の基本的なルールを以下に示す^{10),17)}。3章に記したように、変換後のペトリネットはユーザから見た交換システムの状態遷移を記述したものであるため、プレースへの変換対象となるものは、cpブロックとrsブロックの中の、しかもextやrgといったリソースプロセス内のステートである。

- (1) ステートはプレースに変換する。ただし、必ずしも一つのステートを一つのプレースに変換するとは限らず、一つの安定した状態に含まれるものはそれらのステートをすべて集めて一つのプレースとする。
- (2) ステートとステートの間のインプットおよびアウトプットはそれらを統合して一つのトランジションとする。ただし、データに関するものは検証の対象としないため、タスクは基本的に何にも変換しない。
- (3) SDLではプロセス間のシグナルの送受は非同期通信に基づいており、同期通信を可能とするため

wait ステートを設けることがあるが、これは Fig. 3 に示すようにプレースとはしない。

- (4) プロセスの生成・消滅については、そのリソースを表わすトークンのアイドルプレースを設け、そこからのまたはそれへの状態遷移により表現する。
- (5) インスタンスの概念を有する SDL からその概念のないペトリネットに変換するには、SDL をシミュレートさせる必要があるが、ある安定したステートからの起動は必ず dv ブロックからのシグナルによるものとする。

5.3 変換例

Fig. 1 および Fig. 2 に示したものは、PBX において内線相互接続を中心としたサービスについての SDL 設計仕様である。このうち、3 者会議通話サービスを除いたものを上記変換ルールによって Fig. 4 に示す色付きペトリネット(点線部を除いたもの)に変換する。ここでは、プレートを楕円で、トランジションを太線で表わしている。また、アークの横にその上を通過するトークンの種類とその数を示す。トークンには、本ペトリネットでのリソースである内線加入者と呼出信号の 2 種類のものが存在するが、ここでは簡単のため後者のみに (rg) と付し、前者には特にそれを記さず複数個あるときにその数を示す。なお、2.2 節に記したとおり、Fig. 2 では、Fig. 4 においてアイドル状態からプレース P_5 に至るまでの最も

基本的なサービスについては記していない。

6. 検証例

Fig. 4 のペトリネットについて以下のような検証を行うことができる。

6.1 S インバリエント

Fig. 4 における接続行列を Fig. 5(a) に示す。ここで、内線加入者の変数を X , Y および Z で表わし、その全体集合を U で表わす。なお、Fig. 5(b) は Fig. 4 のペトリネットで色を無色にしたもの、つまり ext や rg の区別をせずにそれらの数だけで表現したものについて求めた接続行列である。Fig. 5(a) の接続行列の S インバリエントを求めると、Fig. 6 に示すように 2 個の S インバリエント、 y_1 および y_2 が得られる。また、これらの S インバリエントから導かれるトークンの保存性について成立する等式を同図に示す。ここで、 $M(P_i)$ はプレース P_i に存在するトークンを示す。これらの列ベクトルのすべてについてその要素が 0 であるプレースは存在しない。つまり、ペトリネットは S インバリエントで被覆されることがわかる。したがって、このペトリネットは有界である。

y_1, y_2 は、それぞれ内線加入者、呼出信号のリソースについて有界であることを示している。また、Fig. 5(b) について求めた S インバリエントからも全く同様の結果

Fig. 4 Petri Nets created from SDL shown in Fig. 2 (a part drawn by dotted lines shows three-way conversation)

$$\begin{array}{c}
 \begin{array}{c}
 \text{A} = \\
 \begin{array}{l}
 T_1 \\
 T_2 \\
 T_3 \\
 T_4 \\
 T_5 \\
 T_6 \\
 T_7 \\
 T_8 \\
 T_9 \\
 T_{10} \\
 T_{11} \\
 T_{12}
 \end{array}
 \end{array}
 \left\{
 \begin{array}{cccccccc}
 -X & X & 0 & 0 & 0 & 0 & 0 & 0 \\
 -Y & -X & Q & -rg & 0 & 0 & 0 & 0 \\
 0 & 0 & -Q & rg & P & 0 & 0 & 0 \\
 Y & 0 & 0 & 0 & -P & X & 0 & 0 \\
 X & 0 & 0 & 0 & 0 & -X & 0 & 0 \\
 X & -X & 0 & 0 & 0 & 0 & 0 & 0 \\
 P & 0 & -Q & rg & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & -P & 0 & P & 0 \\
 0 & 0 & 0 & 0 & P & 0 & -P & 0 \\
 -Z & 0 & 0 & -rg & 0 & 0 & -P & S \\
 0 & 0 & 0 & rg & 0 & 0 & 0 & -S \\
 Z & 0 & 0 & 0 & P & 0 & 0 & -R
 \end{array}
 \right.
 \end{array}$$

$$A = \begin{matrix} & P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 & P_9 \\ \begin{matrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \\ T_{10} \\ T_{11} \\ T_{12} \end{matrix} & \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 1 & 2 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -2 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & -3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 \\ -1 & 0 & 0 & -1 & 0 & 0 & -2 & 4 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 3 \\ 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & -3 \end{bmatrix} \end{matrix}$$

Fig. 5 Incidence matrices of Petri Nets shown in Fig. 4 (excluding three-way conversation)

$$\begin{array}{cccccccccc}
P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 & P_9 \\
y_1 = [1 & 1 & f_{13} & 0 & 1 & 1 & 1 & f_{18} & 1]^T \\
y_2 = [0 & 0 & f_{23} & 1 & 0 & 0 & 0 & f_{28} & 0]^T \\
y_1 \rightarrow M(P_1) + M(P_2) + f_{13} \cdot M(P_3) + M(P_5) + M(P_6) + \\
\quad M(P_7) + f_{18} \cdot M(P_8) + M(P_9) = \Sigma U \\
, \text{ where } f_{13} \cdot (X + Y + rg) = X + Y, \quad f_{18} \cdot (X + Y + Z + rg) = X + Y + Z \\
y_2 \rightarrow f_{23} \cdot M(P_3) + M(P_4) + f_{28} \cdot M(P_8) = \Sigma rg \\
, \text{ where } f_{23} \cdot (X + Y + rg) = rg, \quad f_{28} \cdot (X + Y + Z + rg) = rg
\end{array}$$

$$\begin{array}{cccccccccc}
 & P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 & P_9 \\
 y_1 = & [12 & 12 & 8 & 0 & 12 & 12 & 12 & 9 & 12]^T \\
 y_2 = & [0 & 0 & 4 & 12 & 0 & 0 & 0 & 3 & 0]^T \\
 y_1 \rightarrow & 12 \cdot M(P_1) + 12 \cdot M(P_2) + 8 \cdot M(P_3) + 12 \cdot M(P_5) + 12 \cdot M(P_6) + \\
 & 12 \cdot M(P_7) + 9 \cdot M(P_8) + 12 \cdot M(P_9) = \text{Constant} \\
 y_2 \rightarrow & 4 \cdot M(P_3) + 12 \cdot M(P_4) + 3 \cdot M(P_8) = \text{Constant}
 \end{array}$$

Fig. 6 S-invariants and token-conservation of Petri Nets shown in Fig. 4 (excluding three-way conversation)

$$\begin{array}{l}
\begin{array}{cccccccccccc} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{11} & T_{12} \end{array} \\
x_1 = [1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0]^\top \dots T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_5 \\
x_2 = [1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0]^\top \dots T_1 \rightarrow T_6 \\
x_3 = [1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0]^\top \dots T_1 \rightarrow T_2 \rightarrow T_7 \\
x_4 = [0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0]^\top \dots T_8 \rightarrow T_9 \\
x_5 = [0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1]^\top \dots T_8 \rightarrow T_{10} \rightarrow T_{11} \rightarrow T_{12}
\end{array}$$

6.2 T インバリアント

Fig. 5 の接続行列から Fig. 7 に示すように 5 個の初等的 T インバリエントが得られる。3.1 節で記したように、T インバリエントは基本的にある状態から再びその状態に戻るまでのトランジションの発火回数を示している(発火系列は特定しない)。各 T インバリエントが示す発火系列を同図に示す。したがって、この T インバリエントのうち、 x_1, x_2, x_3 は完結サービスであり、 x_4, x_5 は非完結サービスを示している。たとえば、 x_4 は 2 者通話中からフッキングしてダイヤル中に再びフッキングによってもとの 2 者通話中に戻るものである。この後者のサービスの始まりであるマーキングが、素サービスの中で存在し得るマーキングであることから以下のことがいえる。全トランジションが活性であるためには、ペトリネットが T インバリエントにより被覆されることが必要十分条件である。Fig. 7 に示す T インバリエントのすべてについて、その要素が 0 であるトランジションは存在しない。つまり、トランジションはすべて活性である。

6.3 インクリメンタルな設計

Fig. 2 で、打合せ通話を含む内線相互接続サービスに 3 者会議通話サービスを追加することを考える。

サービス追加時に考慮しなければならないことは、追加するサービスは必ずしも各リソースのアイドル状態から再びアイドル状態に戻るものではなく、すでに作成された仕様でないものだけを追加すればよいことである。つまり、内線相互接続状態から他内線を呼び出しその応答後2者間の通話をしているときに、フッキング操作をした後3者間の会議通話に入る部分だけを追加すればよいのである。SDL レベルでいえば、ext プロセスで2者間の内線相互通話中のステートからフッキングした後、他内線に発信してその応答後、通話に入って再びフッキング操作により3者の会議通話に入る。このとき、シグナルの送受は別としてプロセス図で大きく変更または追加されるのは ext プロセスと cft プロセスである。

この3者会議通話を含むSDLからペトリネットに変換したものがFig.4(点線部を含む)である。たとえば、ここで3者会議に入った後の解放処理(Fig.2で行の右端

が得られ、必ずしも色付きペトリネットでSインバリアントを求める必要のないことが明確になる。

$$\begin{array}{l}
 P_1 \ P_2 \ P_3 \ P_4 \ P_5 \ P_6 \ P_7 \ P_8 \ P_9 \ P_{10} \ P_{11} \\
 y_1 = [1 \ 1 \ f_{13} \ 0 \ 1 \ 1 \ 1 \ f_{18} \ 1 \ f_{110} \ 0]^T \\
 y_2 = [0 \ 0 \ f_{23} \ 1 \ 0 \ 0 \ 0 \ f_{28} \ 0 \ 0 \ 0]^T \\
 y_3 = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ f_{310} \ 1]^T
 \end{array}$$

(a) S-invariant

$$\begin{array}{l}
 T_1 \ T_2 \ T_3 \ T_4 \ T_5 \ T_6 \ T_7 \ T_8 \ T_9 \ T_{10} \ T_{11} \ T_{12} \ T_{13} \ T_{14} \\
 x_1 = [1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T \\
 x_2 = [1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T \\
 x_3 = [1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T \\
 x_4 = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0]^T \\
 x_5 = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0]^T \\
 x_6 = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1]^T
 \end{array}$$

(b) T-invariant

Fig. 8 Two kinds of invariants of Petri Nets shown in Fig. 4 (including three-way conversation)

に“conf. end”と記した部分)が抜けていた場合、ペトリネットに変換するとトランジション T_{14} に相当するものがないため、S インバリエントを求めると P_{10} および P_{11} に相当する要素がいずれも 0 となる。つまり、 P_{10} および P_{11} が S インバリエントに被覆されないため誤りであるとわかる。

また、上記の解放処理が正しく追加されていた場合、新しいペトリネットでも両インバリエントを求めると、Fig. 8 に示すように 3 個の S インバリエントと 6 個の T インバリエントが求められる。これらが、従来から存在するインバリエントを包含することを確認することで、変更前の仕様を正しく引き継いでいることを確認できる。

7. おわりに

交換システムに関する SDL 仕様をペトリネットを用いて代数的に検証する方式について述べた。ソフトウェア開発全体の工程に占めるその重要性から、仕様化段階での検証はきわめて意義が大きいと考える。特に、端末の操作に対する応答という形で示すシステムの状態遷移に的を絞って他言語に変換し、その有界性と活性の検証を行えることを実例をあげて明らかにした。この手法は、基本的に設計者の判断を必要とせず、すべて自動的に行えるので非常に有効であると考え。ここでは、通常のペトリネットとして求めたインバリエントが色付きペトリネットのそれに等しくなっているが、今後それが成立するクラスを明確にしていくとともに、本検証方式の検証範囲の拡大を目指していく予定である。

参考文献

- 1) R. Balzer: Software Technology in the 1990's: Using a New Paradigm, IEEE Computer, 16-11, 39/45(1983)
- 2) CCITT Recommendation: Z. 100: Functional

Specification and Description Language(1988)

- 3) H. Ichikawa, M. Itoh and M. Shibasaki: Protocol-Oriented Service Specifications and Their Transformation into CCITT Specification and Description Language, IEICE Trans. E69-4, 524/535(1986)
- 4) 宗森, 水野: SDL グラフィックエディタの設計と製作, 情報処理学会論文誌, 29-7, 676/685(1988)
- 5) Y. Wakahara, K. Kakuda, A. Itoh and E. Utsunomiya: ESCORT: An Environment for Specifying Communication Requirements, IEEE Trans., SE-6-2, 38/43(1989)
- 6) J. L. Peterson: Petri Net Theory and the Modeling of Systems, Prentice-Hall(1981)
- 7) W. Reisig: Petri Nets An Introduction, Springer-Verlag(1983)
- 8) G. Comparin, G. A. Lanzarone, W. Panzeri and A. Torgano: Analysis of SDL Specifications using Petri Nets Techniques, Proc. of 2nd SDL Forum, 1/12(1986)
- 9) W. Kettunen and M. Lindqvist: Towards Practicality of Predicate/Transition Petri Net Reachability Analysis of SDL, Proc. of 3rd SDL Forum, 285/294(1987)
- 10) 宗森, 武田: ペトリネットによる SDL のシミュレーション機能の検討, 電子情報通信学会, 第 3 回ネット理論研究会, 34/41(1988)
- 11) K. Jensen: Coloured Petri Nets, Petri Nets: Central Models and Their Properties. Advances in Petri Nets 1986—Part I, Lecture Notes in Computer Science, 254, Springer-Verlag, 248/299(1987)
- 12) 長谷川晴朗: 通信ソフトウェア要求仕様化設計へのペトリネットの応用, 計測自動制御学会, 第 9 回離散事象システム研究会, T2(1992)
- 13) G. Memmi and J. Vautherin: Analyzing Nets by the Invariant Method, Lecture Notes in Computer Science, 254, Springer-Verlag, 300/336(1986)
- 14) 上田, 本坊: CP-nets における P-invariant の解法, 電子情報通信学会, 回路とシステム研究会, CAS90-98(1990)
- 15) G. Berthelot and R. Terrat: Petri Nets Theory for the Correctness of Protocols, IEEE Trans. Commun., 30-12, 2497/2505(1982)
- 16) K. Shibata, Y. Ueda, S. Yuyama and H. Hasegawa: A Study on Verification of Service Specification in Communication Software Development, IEICE, E71-12, 1203/1211(1988)
- 17) 田口, 安藤, 長谷川: 中間言語にペトリネットを用いた SDL 仕様設計の一方式, 第 39 回情報処大全, 6S-3(1989)

《付録 A. SDL》

A.1 概要

SDL は、1976 年に最初の勧告されてから旧 CCITT で 4 年ごとに新しい勧告が出されている。SDL は、有限状態機械をモデルとしているが、単に状態遷移だけを表わすのではなく、構造的な概念を有しデータの操作やタスクの処理ができるなど、かなり実際のシステムに適用し得る機能が追加されている。その記法には、コンピュータ処理が容易なテキスト形式の SDL/PR(textual Phrase Representation)と設計者にとって理解の容易なグラフィカル表示の SDL/GR(Graphical Representation)とがあり、基本的に 1 対 1 の相互変換が可能である。なお、SDL は膨大な勧告資料により厳密に定義され

Fig. 9 Hierarchical configuration of SDL

ているが、ここでは説明の都合上そのサブセットである Basic SDL の一部について記す。

A.2 基本概念

SDL は実際の通信システムを設計するうえで重要な構造化の概念を有している。つまり、Fig. 9 に示すように、全体のシステムを複数のブロックに、またブロックを複数のプロセスに静的に分割して、プロセスの中を有限状態機械として表現することとしている。

また、SDL はオブジェクト指向の概念を一部取り入れており、たとえば後述のプロセスのように、定義したタイプに従って1個あるいは複数個のインスタンスを生成し得るものが存在する。

A.3 構成要素

SDL の主要な構成要素を以下に示す。

- (1) システム 全体をシステムと呼ぶ。
- (2) ブロック システムの構成要素である。
- (3) プロセス ブロックの構成要素である。プロセスの定義に対してあらかじめ決められた数までのインスタンスを生成し得る。これをプロセスインスタンスといい、その一つ一つに PId(Process instance Identifier) という、ユニークな番号が付与されている。なお、冗長な表現を避けるため、プロセスインスタンスというべきところをプロセスということが多い。
- (4) スタート プロセスの開始を示す。
- (5) ストップ プロセスを停止する。
- (6) ステート プロセス内の状態を示す。
- (7) シグナル プロセス間の情報の流れを示す。
- (8) インプット 他プロセスからのシグナルの入力を示す。
- (9) アウトプット 他プロセスへのシグナルの出力を示す。

Fig. 10 GR of some PROCESS components

```
PROCESS pr1;
START;
STATE st1;
INPUT sig1;
OUTPUT sig2;
NEXTSTATE st2;
INPUT sig2;
STOP;
ENDPROCESS pr1;
```

Fig. 11 An example of PR and GR

す。

- (10) タスク 変数への値の代入などの処理を実行する。

Fig. 10 にプロセス構成要素の GR 表現を示す。

A.4 簡単な例

A.1 でシステムを階層的にトップダウンにより表現すると記したが、逆に有限状態機械に基づくプロセスがシステムの基本構成要素であるとも考えられる。プロセスの簡単な例の GR 表現と PR 表現を Fig. 11 に示す。なお、このプロセス定義から生成される二つのプロセスインスタンス間で通信が行われることを想定しており、たとえば sig 2 はそれらの間で送受されるシグナルである。

[著 者 紹 介]

長谷川 晴 朗 (正会員)

1972 年、東京大学工学部電子工学科卒業。74 年同大学大学院修士課程修了。同年沖電気工業(株)入社。現在、ネットワークシステム開発センタソフトウェア開発第三部長。通信システムの仕様記述など、通信ソフトウェア開発環境に関する研究に従事。IEEE、電子情報通信学会、情報処理学会などの会員。