社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
SS2002-11 (2002-07)

ペトリネットで記述された簡易ブラウザ型の組込みJavaプログラム動作仕様に対する実行方式の提案

坂上 弘祐† 岡野 浩三† 谷口 健一†

† 大阪大学大学院情報科学研究科 〒560-8531 豊中市待兼山町 1-3
E-mail: †{k-sakaue,okano,taniguchi}@ist.osaka-u.ac.jp

あらまし 簡易ブラウザ型のJ2MEプログラムの仕様に対する実行方式とプログラムの自動導出法を提案する．プログラムの動作仕様をペトリネットで記述することにより，並行・同期制御も容易に表現でき，形式的検証も行いやすい．本稿では，実行制御を表したペトリネット記述から，アプリケーションロジックとリソースを動的にロードし解釈実行するプログラムを導出する．提案方式では，解釈するロジックにペトリネットの接続行列を用い，オフライン実行を考慮した単純な機構を実現することで実行プログラムサイズを抑え，また，通信コストを軽減できる．
キーワード カラーペトリネット，動作仕様記述，J2ME，プログラム導出

A deriving method for J2ME programs on personal appliances from application descriptions in Petri-nets and their executional environment

Kousuke SAKAUE†, Kozo OKANO†, and Kenichi TANIGUCHI†

† Graduate School of Information Science and Technology, Osaka University Machikaneyama 1-3, Toyonaka-shi, 560-8531 Japan
E-mail: †{k-sakaue,okano,taniguchi}@ist.osaka-u.ac.jp

Abstract This paper presents a method automatically deriving J2ME programs from application descriptions in Petri-nets, which can easily express parallel and concurrent actions, and can be analyzed by means of formal analysis methods. The derived program executes a partial sequence of whole transitions downloaded from a peer server. The sequence and related resources are downloaded in step-by-step manner according to matrix equations of the given Petri-net. The proposed method can reduce the size of a client-side program and also can reduce the cost of communication.
Key words coloured Petri-nets, whole system description, J2ME, code generation

1. まえがき

ペトリネット（PN）を用いたシステムのモデリングあるいは実装の手法が有用な設計支援法として提案されている[1]～[3]．ソフトウェアの設計においては，設計者の負担を軽減し，大幅な効率の向上が行える設計支援が望まれる．このための手法の1つとして，抽象度が高い動作仕様を用いて設計を行い，実装プログラムをその動作仕様から自動的に導出する手法が有用である．また，記述する動作仕様に設計検証を形式的かつ自動的に行えるモデルを用いることで高信頼設計が見込まれる．著者らはこの点をふまえて，高レベルPNとして知られるカラーペトリネット（CPN）[4]で記述されたネットワークアプリケーション動作仕様からJavaプログラムを自動導出する手法を提案した[5]．

本稿では，近年普及しつつある携帯・組込みJava（J2ME）を対象とし，簡易ブラウザ型アプリケーション[6]のCPN動作仕様記述からの導出方法を新たに提案する．能力や資源が限定された実行環境である携帯・組込みJavaでは，アプリケーションのファイルサイズに大きな制限があり，複雑な処理を行うプログラム全体を実装することは難しい．そこで，アプリケーションロジック（動作命令）とリソース（データ）をネットワークから動的にロードして解釈を行う方式を採用し，それに基づいたブラウザ型アプリケーションの仕様に対する実行方式を提案する．アプリケーションの実行のための基本ルーチンは携帯端末側にすべて実装する．携帯端末側で実行を行うアプリケーションの実行制御をPNで表現した動作仕様として記述し，そこから，クライアントサイド（携帯端末）とサーバサイドの実行プログラムを自動導出する．

提案する実行方式では，動作命令を逐次解釈し実行する際に，PN の接続行列と状態方程式を用いる．これにより，クライアントの解釈実行部が単純な機構となり，プログラムのファイルサイズを抑えることができる．実際に導出されるプログラムにおいては，クライアント側のプログラムサイズを小さく抑えた上で，なるべく，サーバクライアント間の通信コストを抑える工夫も必要である．また，クライアント側で実行できる動作はクライアント側でなるべく実行すべきである．このような制約のもとでよい実行プログラムを導出するために，以下のような通信上の工夫を行っている．

（1） サーバ側はアプリケーションの動作に必要と予想される部分的な実行系列のデータをなるべくまとめて動作命令列としてクライアントに送信する．

（2） 導出にあたっては，一度に送信可能な実行系列のデータとそれに必要となるアプリケーションデータを，動作仕様とアプリケーションデータに関する情報および送信データの最大サイズから静的に決定し，それをもとに最終的な送受信データを決定する．

以後，2. では動作仕様記述について述べる．3. ではプログラムの導出について説明する．4. では適用例題について述べ，5. でまとめる．

2. 動作仕様記述

2.1 ブラウザ型アプリケーション

携帯・組込み Java プログラムは能力や資源が限定された実行環境を対象としており，携帯電話等のクライアント側で提供されている処理（API 等）や実行時の CPU の処理能力などに制限がある．とりわけアプリケーションのファイルサイズ（J2ME クラスファイルとリソースファイル）には，大きな制限がある．

これらを考慮すると現時点ではクライアント側で複雑な処理をまとめて行うことは困難である．そのため，ブラウザ型アプリケーション [6] として実装するのが 1 つの解である．ブラウザ型アプリケーションとは，Web ブラウザのように，コンテンツをネットワークからロードして再生する実行方式をとるような構造を持つアプリケーションである．

このブラウザ型アプリケーションでは，実行するために必要な最小限の基本ルーチンを携帯端末側に実装し，場面に応じたアプリケーションロジックとアプリケーションリソースをネットワークから動的にダウンロードして解釈実行する．この実行方式では，携帯電話等のように無線によって常時ネットワーク接続されているとみなすことができる環境では，基本ルーチンの対応可能な範囲においてサイズに依存しないアプリケーションを作ることが可能である．カードゲームを例としてあげると，基本的な命令（カードを配る，カードの表示，カードの選択など）が与えられていれば，ゲームのバリエーションを多く持たせることができる．

2.2 ペトリネット記述によるアプリケーションの動作仕様

ここでは，PN を用いたブラウザ型アプリケーションの動作仕様記述について述べる．提案する自動導出法では，CPN デザインツール Design/CPN [4] で記述した動作仕様から，プログラム群を導出する．アプリケーションの動作仕様を UML [2] や State Chart [7] ではなく PN で記述することにしているが，これによって記述能力が限定されるわけではない．PN のネット全体をアプリケーション全体，サブネットをアプリケーションの下位構造（基本ルーチン部に相当），マーキングを各状態，トランジションを状態からの遷移イベントとすれば，十分に表現能力があるといえる．

動作仕様ではアプリケーションの実行制御を記述する．カードゲーム仕様を表した記述の一部である図 1 を例として説明する．

トランジションが全体のシステム上での遷移イベントを表す．クライアント側で行いたい処理（基本ルーチン）はそれに対応するトランジションにラベルとして与えられる．そのトランジションは後で説明する階層構造を持ち，一般に部分 PN に展開される．そのようなトランジションは図中で HS と表示されている．プレースにはシステムの実行制御の状況を表すプレースと，アプリケーションリソースのバッファを表すプレースがある．実行制御の状況を表すプレースと遷移イベントを表すトランジションを用いて全体のシステムの動作の流れを PN による動作仕様として記述する．右下に FG と指定されているプレースがアプリケーションリソースのバッファを表し，このプレースが保持するトークンがアプリケーションリソースを表す．アプリケーションリソースのプレースと遷移イベントを表すトランジションをアークで接続することにより，イベントとリソースを関連づける．

基本ルーチン部にあたる下位構造記述の例を図 2 で示す．基本ルーチン部では実行制御の状況を表す入出力プレースとは別に基本ルーチン内で使用したいアプリケーションリソースを表すプレースも記述し，記述内のイベントを表すトランジションと双方向アークで接続する．このようにして，基本ルーチンの実行時に参照するパラメータとしてアプリケーションリソースを設定できる．

この例は，ある文字列を表示するための基本ルーチンの記述を表している．制御を表す入力プレース setup とは別に文字列データバッファを表すプレース label が記述され，イベントを表すトランジションと関連づけすることにより，実行時に文字列データバッファをパラメータとして参照することを表している．

最終的に，アプリケーションのシステム全体を表す動作仕様記述の階層構造（図 3 ）がデザインツールから自動的に得られる．階層構造の上位にアプリケーションの動作仕様を記述したページ，階層構造の下位にクライアントの基本ルーチンを表すページがある．別途与える変数宣言のページも含め，PN による動作仕様記述は主に 3 つのページ内容で構成される．

入力となる PN に対する制約は以下のとおり．

- 各部分ネットとして自由選択ネット [8] を使用する．基本ルーチン部記述は 1 シンク．
- アプリケーションリソースプレースとトランジションを結ぶ両方向アーク以外は自己ループ構造を持たない．
- 制御に用いるトークンの型は論理型・整数型・実数型・文字列型・列挙型のみ．

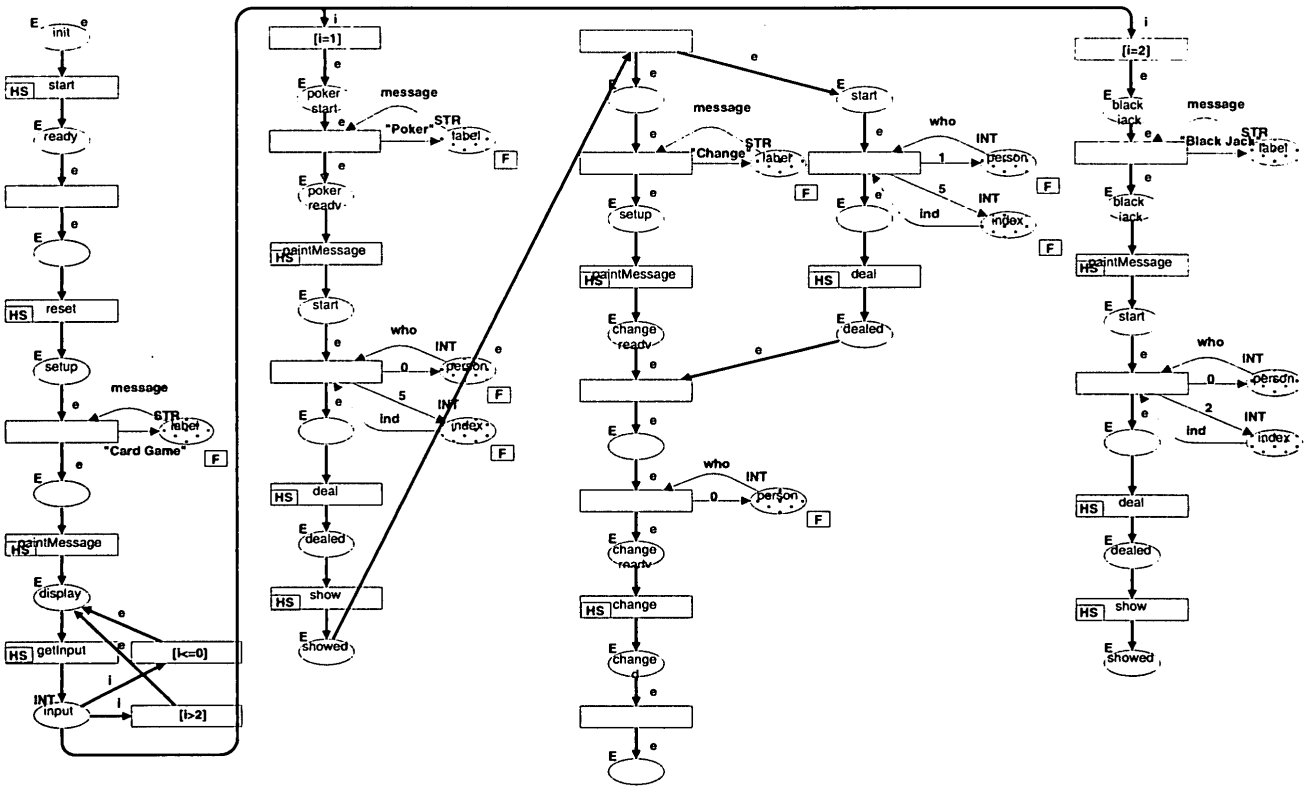

図 1　PN 動作仕様のアプリケーション記述

Fig. 1　PN page modelling a J2ME application

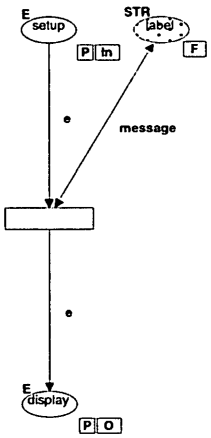

図 2　PN 動作仕様のメッセージ表示イベント記述

Fig. 2　PN page modelling a paint-message event handling

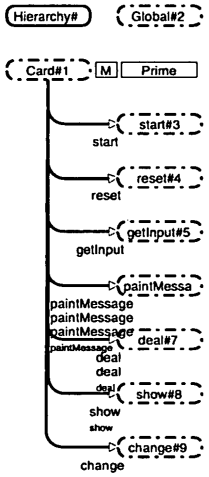

図 3　PN 動作仕様の階層記述

Fig. 3　Hierarchical structure of PN

2.3　動作仕様から導出されるプログラムの実行形式

PN による動作仕様記述が与えられたときに，クライアントサイドとサーバサイドの実行プログラムを自動導出する．この方式において以下の点を考慮に入れる必要がある．

- Java のオフライン使用
- 通信環境の制限

アプリケーションを Java で実行する必然性の 1 つにオフラインでの使用が挙げられる．すなわち，サーバとなるべく通信しなくてもクライアント側だけでアプリケーションが動作できることに意味がある．また，クライアントとサーバの通信は HTTP（もしくは HTTPS）通信のみであり，サーバからクライアントへの接続ができないという制限がある．また，サーバに全て実行を委せるアプリケーションにすると，毎回クライアントからサーバにデータをロードすることが必要となり，接続が少しでも途切れた場合は，アプリケーションが動かなくなってしまう．

本稿では上記を考慮し，サーバが動作命令列をクライアント側に渡し，クライアントではそれを逐次実行していく実行方式を実装するクライアントプログラムを導出する．実装するシステムの構成を図 4 に示す．与えられた動作仕様記述から，図 4 のようなクライアントとサーバのプログラムを導出する．クライアント側には，基本ルーチンを逐次実行するインタプリタ部分に加え，動作命令列やデータをロードするための通信部を組

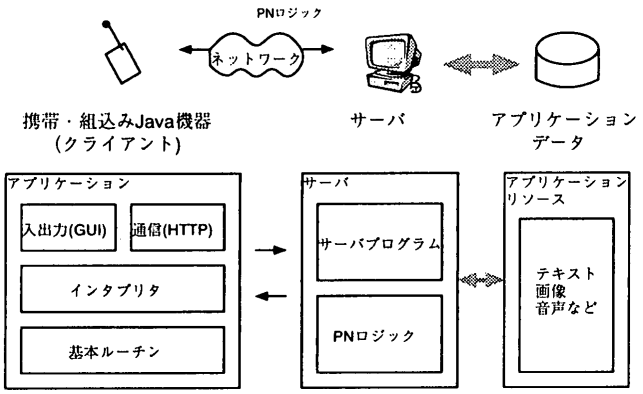

図 4　システムの構成
Fig. 4　Overview of the system

み込む．必要に応じて入出力部分も導出する．また，アプリケーションで必要となる基本ルーチン部分は，そのインタフェースとなる関数宣言部のみを導出する．サーバ側には，クライアントの通信部から送られてくる要求を解析し，必要なデータを渡すプログラムと，解析のために必要な PN の動作命令全体を組み込む．さらに，テキストコンポーネント（文字列や数）や画像などのアプリケーションデータはすべてサーバ側に保存しておき，クライアント側が必要時にサーバからバイナリデータとしてダウンロードする．

最終的に自動導出されたクライアント側のプログラムに対して，設計者は基本ルーチンの中身だけを実装するだけでよい．すなわち，アプリケーションの実行制御のみを動作仕様として PN で記述しさえすれば，先程まで述べてきた細かい制御や送受信を意識することなく設計できる．

3. プログラム導出

3.1 あらまし

クライアント側のプログラムは，上記の実行方式を実現するために，動作命令列とアプリケーションデータをサーバからロードし解釈実行する必要がある．そのため，基本ルーチン部分に加えインタプリタ部と通信部を別途持っている．

インタプリタ部は，通信部を介して得られた動作命令列を逐次判定し，その動作命令（イベント）が実行可能（発火可能）であるならば，現在の状態から遷移を行い，必要に応じて動作命令に対応した基本ルーチンを実行する．これを実現するため，クライアント側には現在の状態（カレントマーキング）を保持しておき，サーバ側はクライアント側の要求に応じて，実行（発火）できそうなイベント（トランジション）の集合から，実行できる条件（発火可能条件）を判定できるような動作命令列をクライアント側に送信する．

3.2 詳　　細

インタプリタ部には動作命令列を逐次解釈し実行していくスケジューリングアルゴリズムを与える．このアルゴリズムは，文献 [9] で紹介されている PN のシミュレーションにおいてトランジションの発火判定を決定するためのスケジューリングアルゴリズムを新たに拡張したものである．

このアルゴリズムは，あるトランジションが発火した場合，その発火によって影響を受ける（すなわち発火可能になるかもしれない）トランジションは，発火したトランジションの近傍トランジションだけという原則に基づいている．そこでトランジションの集合を発火可能でないトランジションの集合 $T_{disabled}$ と，発火可能か未定のトランジションの集合 $T_{unknowns}$ の 2 つの集合に分ける．発火可能かどうかの判定は $T_{unknowns}$ の集合のみで行い，もし発火可能でなければそのトランジションを $T_{disabled}$ の集合へ移動させ，発火可能であればそのトランジションと近傍トランジションを $T_{unknowns}$ の集合に移動させる．従来のスケジューラではすべてのトランジションを 1 つのリストに格納し，そのリストを循環して発火可能かどうかの判定を行うトランジションを決定している．この方針では，非決定性選択の場合にリストに格納されている順番によって片方のみが常に発火し続けるという偏りが起こりうる．本稿では，$T_{disabled}$ と $T_{unknowns}$ を用いたアルゴリズムを用いることでその問題が解消している [9]．さらなる拡張として，この一連の作業をクライアントとサーバに分離する．アルゴリズムにおいてクライアント側が解釈実行を行うために必要なものは $T_{unknowns}$ の情報のみである．具体的には，負荷が大きいトランジションの集合の管理と必要な演算をサーバで行い，クライアントは $T_{unknowns}$ のみを動作命令列としてサーバから取得し，解釈実行する．クライアントはサーバから取得した動作命令列を解釈したアプリケーションの実行がすべて終了するとその結果と関連データをサーバに渡し，サーバは結果からクライアントの動作と現在の状態を把握し，再び次の動作命令列を渡すために必要な $T_{unknowns}$ を決定する．

アルゴリズム 1：インタプリタ部の動作命令解釈スケジューリングアルゴリズム

T, set of transition instances in the given PN discriptions

$T_{disabled} \Leftarrow \emptyset$, subset of T with elements having status *disabled*

$T_{unknowns} \Leftarrow T$, subset of T with elements having status *unknown*

$M \Leftarrow M_{init}$, current marking

$step \Leftarrow \emptyset$, step counter

while make steps until deadlock or until a break is requested **do**

  while $T_{unknowns} \neq \emptyset$ **do**

    $t_{candidate} \Leftarrow \langle\langle$ random element from $T_{unknowns} \rangle\rangle$

    $status \Leftarrow \langle\langle$ Try finding an enabled binding for $t_{candidate}$ and M , make it occur and return *occurred*. Otherwise return enabling status. $\rangle\rangle$

    if $status = disabled$ **then**

      $\langle\langle$ move $t_{candidate}$ from $T_{unknowns}$ to $T_{disabled} \rangle\rangle$

    else if $status = occurred$ **then**

      $step \Leftarrow step + t_{candidate}$

      $T_{dependents} \Leftarrow (t_{candidate} \bullet) \bullet$

      $\langle\langle$ move $T_{dependents} \cap T_{disabled}$ from $T_{disabled}$ to

$T_{unknowns}$ ⟩⟩

end if

end while

if *step* ≠ ∅ **then**

$T_{unknowns}$ ⇐ ⟨⟨ refresh from *step* ⟩⟩

end if

end while

以上のスケジューリングにより逐次解析する動作命令を決定し，発火可能状態を判定している．発火可能条件を解析するための動作命令として，PN でモデル化されたシステムの性質を表す接続行列と状態方程式を用いる [8].

接続行列 $A = [a_{ij}]$ は n 個のトランジションと m 個プレースを持った PN に対して与えられる $n \times m$ の整数行列であり，各成分は次の式 (1) で与えられる．

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-} \tag{1}$$

ここで，$a_{ij}^{+} = w(i,j)$ は，トランジション i からその出力プレース j へ向かうアークの重みであり，$a_{ij}^{-} = w(j,i)$ は，トランジション i の入力プレース j からトランジション i へ向かうアークの重みである．

トランジション i の発火可能条件は次式 (2) で表すことができる．

$$a_{ij}^{-} <= M(j), j = 1, 2, \ldots, m \tag{2}$$

さらに，トランジション i が発火可能であった場合に，次の状態を表すマーキング M' は現在のマーキング M から次の状態方程式 (3) で容易に求められる．

$$M'(j) = M(j) + a_{ij}, j = 1, 2, \ldots, m \tag{3}$$

インタプリタ部はこの式の判定処理部を持つ．すなわち，クライアント側では現在の状態であるマーキング M だけを保持しておき，サーバ側は $T_{unknowns}$ のすべてのトランジション i の成分である a_i^{+} と a_i^{-} のみを動作命令としてクライアント側に渡すことで，逐次実行が可能となる．インタプリタ部の導出は，マーキングの値などに影響することになるが，与えられた動作仕様から静的に決定できる．

最後に発火可能であることが判定できれば，動作命令で指定された基本ルーチン内の命令を実行する．すなわち，対応するイベント（トランジション）が生じた（発火）時に，イベント通知されて実行されるイベントリスナ（命令）を持つ簡易転送イベントモデルをプログラムの基本ルーチンの実行部分として導出する．

アプリケーションの基本ルーチン部分は Java のメソッドとしてインタフェース部分となる関数宣言部を導出する．インタフェース部分には呼び出される実行命令と動作仕様記述内で指定したアプリケーションリソースをパラメータとして参照している．

例として図 3 から得られるプログラムのメソッドは以下のとおりである．

例 1：導出される基本ルーチン部の例

```
public void paintMessage( String message ) {
    /** 注釈・ここに基本ルーチンの詳細な実行を記述 **/
}
```

この例ではある文字列を表示するための基本ルーチンの記述を表していたが，その実行命令として paintMessage メソッドが与えられており，文字列データバッファを関連づけしていたことにより，文字列データバッファをメソッドのパラメータとして参照している．

このメソッドの呼出しはインタプリタ部が動作仕様記述から得られた動作命令列を解釈し動的に行われる．

上記の方針で，与えられた動作仕様記述からインタプリタ部分と基本ルーチンのインタフェース部分が導出される．設計者は最終的に基本ルーチンの中身だけ実装することで，実行を制御する本体やアプリケーションリソースのロードなどファイルサイズを緩和するための実装の詳細を特に意識する必要なくアプリケーション全体を実装することができる．実際に設計者が記述する基本ルーチンの中身の例を示す．以下の例は文字列データをウィンドウに表示するコードを表している．

例 2：実際の記述例

```
public void paintMessage( String message ) {
    // ウィンドウを作成
    Panel panel = new Panel();
    // ラベルに貼る文字列を設定
    Label label = new Label( message );
    // ウィンドウにラベルを追加
    panel.add( label );
    // ディスプレイに表示するカレントウィンドウの指定
    Display.setCurrent( panel );
}
```

3.3 効 率 化

今まで述べてきた実行方式により，通信環境を生かしたブラウザ型アプリケーションを実装できる．しかし，現在のところ無線通信の設備環境や利用料金の関係から通信を頻繁に使用することは難しい．そこで，導出されるブラウザ型アプリケーションにおいて，逐次ロードするだけではなく，できる限り通信量や回数を少なくすることが望まれる．クライアント側のプログラムは動作命令列を受け取る際にサーバ側と通信を行うがその際に以下の点を考慮に入れる必要がある．

- サーバと通信する時に，複数の要求をまとめる．
- ある程度動作命令を先読みし，動作命令列を得る．
- 携帯・組込み Java の実行環境にあるメモリを使用し，動作命令やデータをキャッシュする．

これらの問題に対して例えば，サーバで決定している $T_{unknown}$ を数ステップ分先読みし，クライアントに渡す方法が考えられる．

サーバは現在の状態から次状態の $T_{unknown}$ を決定するが，その $T_{unknown}$ から順次幅優先で数ステップ分先読みすることで，$T_{unknown}$ の候補を許す限り決定する．

アルゴリズム 2：サーバの $T_{unknown}$ 決定アルゴリズム

for $T_{unknowns}$ **do**

$t_{candidate} \Leftarrow \langle\langle$ a element from $T_{unknowns} \rangle\rangle$

if $t_{candidate}$ *inSubnet* $\wedge \bullet t_{candidate} =$ fusion place **then**

if $\bullet t_{candidate}$ *clientside* $\vee \bullet t_{candidate}$ is *refresh* **then**

$applicationdata \Leftarrow \bullet t_{candidate}$

end if

end if

$T_{nexts} \Leftarrow (T_{unknown} \bullet) \bullet$

$\langle\langle$ move $T_{nexts} \cap T_{disabled}$ from $T_{disabled}$ to $T_{unknowns} \rangle\rangle$

end for

送信する実行系列の候補を決定した後，送信データの最大サイズと一度に送信可能な実行系列のデータを，動作仕様とアプリケーションデータに関する情報から決定する．また，サーバ側が保持しているアプリケーションデータにはそれぞれフラグを付与することでデータの状態を監視する．クライアントへ渡す必要があるデータをフラグから決定し，必要となるアプリケーションデータも同時に送信する．

4. 適用例題

以上のアルゴリズムに基づく導出系のプロトタイプを実装した．この導出系は，PN を入力とし，J2ME プログラムの導出を行うトランスレータである．入力は Design/CPN によって得られる XML テキストデータファイルを使用する．XML parser(Xerces Java Parser 1.4.4) を利用し，Java(Java 2 SE 1.3.1) で実装した．

例題として簡単なブラウザ型アプリケーションのコンテンツを考え，実装した導出系に適用する．

なお，今回 J2ME の実験対象として 503Java（導出されるプログラムは CLDC+ドコモ拡張）を使用している．

4.1 オンライン問題集

TOEIC のような問題集を例にとる．ある程度の基本命令が与えられていれば，多い状態数（問題数）も表せることが可能である．基本命令としては，問題や選択肢を表示する，選択肢から結果を受け取る，出題のタイムリミットをカウントするなどが挙げられる．

- 記述するアプリケーション オンライン問題集：60 問
- PN 記述
  - ノード（プレース，トランジション）数：約 900
  - コネクタ（アーク）数：約 600
  - 下位構造記述（基本ルーチン）数：10
- 導出されたファイルサイズ 約 6KB
- 導出時間 約 12 秒

4.2 カードゲーム

カードゲームは基本状態（カードの枚数など）やそれらを処理する基本命令は一意に定まる．この基本命令からゲームの種類を複数生成することができる．基本命令としては，カードを配る，カードの表示，カードの選択などが挙げられる．

- 記述するアプリケーション：10 種類のカードゲーム
- PN 記述
  - ノード（プレース，トランジション）数：約 600
  - コネクタ（アーク）数：約 600
  - 下位構造記述（基本ルーチン）数：9
- 導出されたファイルサイズ 約 4KB
- 導出時間 約 6 秒

(PentiumIII 500MHz, メモリ 512M, Linux)

5. あとがき

本稿ではペトリネットで記述した簡易ブラウザ型の J2ME プログラムの動作仕様に対して，アプリケーションリソースと動作命令列をサーバから逐次ロードし実行する方式を提案し，自動導出する方法を述べた．

提案した実行方式により，能力や資源が限定された実行環境である携帯・組込み Java 側のプログラムのファイルサイズを抑えた上で，より複雑な処理の実行が可能となる．導出されるプログラムに，ペトリネットの接続行列と状態方程式を用いて単純な機構として実現した解釈実行部分を実装することで，サーバとなるべく通信せずに実行が行うことができ，通信コストを軽減できる．また，プログラムを自動導出することで，実装における時間や負担の軽減が期待できる．

今後の課題として，評価実験を行なうこと，生成コードの効率化や簡略化などが挙げられる．

文　献

[1] Kjeld H.Mortensen, "Automatic Code Generation Method Based on Coloured Petri Net Models Applied on an Access Control System," ICATPN2000, LNCS1825, pp. 367–386, 2000.

[2] Mohammed Elkoutbi and Rudolf K.Keller, "User Interface Prototyping based on UML Scenarios and High-level Petri Nets," ICATPN2000, LNCS1825, pp. 166–186, 2000.

[3] 山口昭，岡野浩三，谷口健一，"時間制約付きカラーペトリネットで記述されたワークフローからのスケジュール導出，" 電子情報通信学会技術研究報告，Vol. 101, No. 629, pp. 23–30, 2002.

[4] Kurt Jensen, "Coloured Petri nets Vol.I, Vol.II, Vol.III," Springer-Verlag, 1992.

[5] 坂上弘祐，岡野浩三，谷口健一，"カラーペトリネットで記述されたネットワークアプリケーション動作仕様からの Java プログラムの自動導出，" 2002 信学全大，分冊 1, pp. 151–152, 2002.

[6] 鷲見豊，"プログラミング i モード Java，" オライリー・ジャパン，東京，2001.

[7] Pei Hsia, Jayarajan Samuel, Jerry Gao, David Kung, Yasufumi Toyoshima and Cris Chen "Formal Approach to Scenario Analysis," IEEE Software, pp. 33–41, 1994.

[8] 村田忠夫，"ペトリネットの解析と応用，" 近代科学社，東京，1992.

[9] Kjeld H.Mortensen, "Efficient Data-Structures and Algorithms for a Coloured Petri Nets Simulator," MOCA01, pp. 57–74, 2001.