ソフトウェア工学 99−13
（1994．7．8）

適応型ユーザナビゲーション機構とその評価

白 貞元 † 平原 厚志 † 深澤 良彰 †
† 早稲田大学理工学部

アプリケーションの使用方法の説明には，オンラインヘルプが用いられることが多い．しかし，通常のオンラインヘルプは，実行したい機能の操作方法を知るためには，多くの情報の中から必要な情報を探さなければならないという問題点がある．我々は，ユーザに必要最小限の情報を提供することを目的とした研究を行なってきている．本研究では，ある目的を達成するために必要となる情報をユーザに提供して操作の誘導を行なうことをナビゲーションと呼ぶ．効率良いナビゲーションのために，現在のアプリケーションの状態を反映し，ユーザが既に知識を持っている操作に関する情報は表示しないナビゲーションシステムを提案する．

このようなナビゲーション機能をもったアプリケーションシステムとして，複雑なグラフィカルユーザインターフェイス (GUI) をもったものを考える．GUI アプリケーションの構造は，イベント駆動型であることが多い．そこで，本手法では，そのモデル化に適しているペトリネットを用いて，GUI アプリケーションの構造を表す．そして，そのペトリネットから，ナビゲーションシステムおよびアプリケーションプログラムを作成する．この両者がペトリネットを通じて情報交換をすることにより，アプリケーションの状態を反映したナビゲーションが可能となる．

本稿ではユーザに対する適応化の機能とその評価について述べる．

An Adaptive User Navigation Mechanism and its Evaluation

Jeongwon Baeg† Atsushi Hirahara† Yoshiaki Fukazawa†
† School of Science & Engineering, Waseda University

Recently, there have been many on-line help systems that provide usage explanations for applications. However, they have some problems: it is difficult to find out necessary information in many help messages and to understand how to obtain complete information. We have studied a navigation mechanism which can provide a user with essential information and guide a user so that he can achieve an intended action. In this paper, we propose a navigation system which does not display navigation information on decisions that a user has already made by reflecting the present state of an application.

An application system with complex GUI is used for realizing our navigation mechanism. Most GUI applications' structure is event-driven. Therefore, in order to describe the structure of an application, Petri nets which are suitable for modeling event-driven systems are used in our research. An application system and its navigation system are constructed based on these Petri nets. By communicating between an application system and its navigation system through these Petri nets, user navigation system reflecting the execution state of an application can be realized.

In this paper, we describe an adaptive navigation mechanism for a user and its evaluation.

1 はじめに

アプリケーションの使用方法の説明に，オンラインヘルプが用いられる場合が多くなってきている．既に，ハイパーテキストなどを用いたオンラインヘルプが実用化されている．オンラインヘルプはペーパーマニュアルに比べて，次のような利点があるといわれている [1].

- アプリケーション実行中に参照できる
- 迅速に情報を引き出せる

しかし，現状のオンラインヘルプは，次のような問題点をかかえている．

- 必要な情報と，そうでない情報が混在している
- 情報を引き出していく手順が判りにくい

各種のオンラインヘルプの内容を検討してみると，次の3種類に分類できることがわかる．

- その機能の概要の説明 (指定方法を含む)
- オプショナルな機能とその指定方法
- その機能を実行するために，予め実行しておかなければならない機能の指定

本稿では，この最後の内容に注目する．すなわち，ある目的を達成するために必要となる情報をユーザに提供して操作の誘導を行なうことをナビゲーションと呼ぶ．そして，ある機能を実行するためには，どの機能を実行しておかなければならないかを意味する結果をナビゲーション情報と呼ぶ．また，このようなナビゲーション情報を表示するシステムをナビゲーションシステムと呼ぶ．通常，ナビゲーションという用語は，次にどのような作業をすれば良いかを示すものを意味するが，ここでは，ある作業をする前にどのような作業をすれば良いかを示す点が異なる．このナビゲーション情報は，アプリケーションに対して，順を追って指示を行なっていくような場合に特に有効である．

本稿では，前述のオンラインヘルプがもつ問題点を解決するため，このナビゲーション情報を表示するシステムの機構についての提案を行なう．

不要な情報を取り除き，的確なナビゲーション情報を提供するためには，そのユーザのそのアプリケーションに対する理解度を反映させることが重要である．本研究では，この理解度として，その時点のアプリケーションの状態を考慮する．すなわち，ユーザが既に行なった意志決定についてはその知識を持っているという非常に単純なユーザの理解モデルを構築し，これに従ってナビゲーション情報を表示する機能を提案する．

一方，効率的にナビゲーション情報を引き出すためには，グラフィカルユーザインタフェース (GUI) の研究・実用化が盛んに行なわれてきているので，これを積極的に利用することとする．しかし，複雑な GUI の構築は困難であるため，これに対する何らかの解決策が必要である．

複雑な GUI を効率的に設計・実現するためには，GUI 向きのモデルを用意し，利用することが有効である [2]. このためのモデルとして，状態遷移モデル [3]，ペトリネットモデル [4][5]，メッセージシーケンスチャート (MSC) モデル [6] などいくつかのモデルが提案されている．これらの中で，ペトリネットは，視認性，シミュレーション可能性，数学的な検証容易性などにすぐれている．

我々は，これらの特徴に着目し，基本モデルとしてペトリネットモデルを採用した．ただし，より GUI を記述するのに適するようにいくつかの拡張を行なっている．

本研究で提案する機構では，まず，GUI を含んだアプリケーションの構造を表現するペトリネットを作成する．このペトリネットをもとにアプリケーションプログラムを開発する．この際，ペトリネットを介して，GUI アプリケーションとナビゲーションシステムは通信を行なうようにする．即ち，ユーザがアプリケーションに対して行なった操作による状態の変化は，このペトリネットへと反映される．一方，ナビゲーションシステムは，このペトリネットの状態に従いながら，ユーザから要求されたナビゲーション情報を表示する．この機構を容易に実現するために，状態の遷移をナビゲーションシステムに伝えるようなプログラムの断片をペトリネットから自動生成する．開発者はこれに手を加えて，全体の GUI アプリケーションを完成させる．

本稿では，まず本手法のもつ特徴について述べた後，各構成要素について述べる．続いて，その一部であるナビゲーション情報の導出機構と，アプリケーション開発の支援について述べる．最後に，その評価について言及する．

2 本システムの特徴

2.1 冗長なナビゲーション情報の削減

本システムはユーザに提示すべきナビゲーションシステムの出力をアプリケーションの状態によって変化させ，ユーザにとって必要かつ十分な情報によりナビゲーションを行なう．つまり，冗長な部分を取り除いた適切な情報をユーザに提供することを目的としている．

例えば，音声を録音／再生するアプリケーションを考える．録音機能を操作するためには，あらかじめ，「録音するファイル名をセットする」および「録音ボタンを押す」という機能が実行されていなければならないとする．ユーザが録音機能のナビゲーションを要求した時点で，既にファイル名が設定されていたとすると，このユーザに対するナビゲーション情報としては，録音ボタンに関する情報だけで十分である．すなわち，ファイル名の設定に関する情報は冗長であり，表示しない．

2.2 アプリケーションの状態の反映機構

アプリケーションの状態は，常にナビゲーションシステムに反映されなければならない．そのためには，両者の間で通信を行なう必要がある．つまり，アプリケーションプログラムの内部に状態の検出を行なう機構を組み入れなければならない．また，ナビゲーションシステムとアプリケーションの間で整合性が保たれている必要がある．

本手法では，アプリケーションの構造をペトリネットで表し，これを用いてナビゲーションシステムを制御する．まずアプリケーション開発者は目標とするアプリケーションの構造を表すペトリネットを作成し，それを元にナビゲーション情報およびアプリケーションのプログラムを作成す

る．アプリケーション実行時には，その状態の変更は，このペトリネットへと反映され，ナビゲーション情報はこのペトリネットの状態に基づいて表示される．

2.3 GUIアプリケーションの開発支援機能

開発者は，本システムが提供するエディタを利用してペトリネットの作成と同時にそのペトリネットの構造を参照しながらナビゲーション情報を編集することができる．また，作成されたペトリネットを元に自動生成された一部分のプログラムに手を加えてアプリケーションプログラムを完成させる．ペトリネットの作成段階で，システムの挙動のシミュレーションを行うことは，一種のプロトタイピングと捉えることができる．このエディタは，ペトリネットのシミュレーション機能をもっている．ペトリネットの動作を見ながら修正を繰り返すことは要求分析設計作業の大きな助けとなる [7][8].

このようにして作成したペトリネットから，ナビゲーション情報導出に必要な情報と，アプリケーションの構造および駆動機構を生成する．つまり，ペトリネットからアプリケーションのプログラムの一部分を自動生成することにより，プログラム作成コストの削減，および，アプリケーション本体とナビゲーションシステムとの整合性の保持を実現している．

2.4 GUIアプリケーションの表現支援機能

本システムでは，GUIアプリケーションの記述を容易にするための新たな記法を導入した．GUIアプリケーションにおいては，多くのウィンドウ，メニュー，ボタンなどが使用される．このようなGUI部品の特性や，制約を容易に記述できるようにした．

例えば，GUIアプリケーションでは，一つの機能を実現するために複数の手段が可能である．又，メニューやボタンの選択により，多数に分岐することも多い．このような場合に備えて，専用の表現形式を用意している．

3 本手法の構成要素

本手法において実現されるソフトウェアの構造を図1に示す．図1の破線より上側は，ナビゲーションシステムを備えたGUIアプリケーションを表している．一方，下側は本手法で作られるアプリケーションを構築するためのエディタである．

個々の機能の実行結果が影響するアプリケーションの状態と，対話部品および機能はナビゲーションシステム内にあるペトリネット上に表わされている．ナビゲーションシステムは，このペトリネットの状態（マーキング）に従い，ユーザの要求するナビゲーション情報を出力する．

4 ペトリネットによる表現

4.1 ペトリネットへの基本的対応付け

ペトリネットはプレース，トランジション，アーク，マーキングによって定義される．本手法ではペトリネットの各要素に対して，GUIアプリケーションにおける構成要素を以下のように対応付ける．

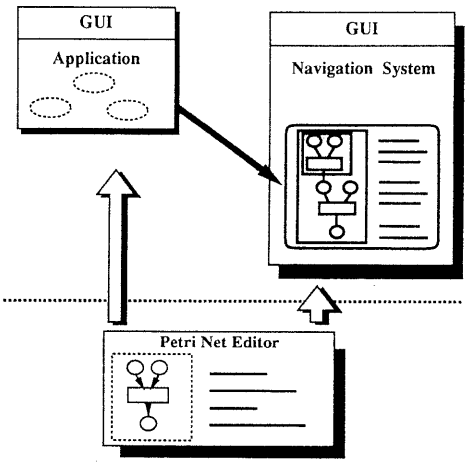


図1: 本手法における構成要素

- プレース（条件）：アプリケーションの局所的な状態

  あるトランジションに対する入力プレースは，トランジションが発火するための条件となるので，アプリケーションの機能を実行するための条件となる状態とする．また，出力プレースはアプリケーションのある機能を実行した結果の状態と定義する．

- トランジション（事象）：アプリケーションの機能及び対話部品

  トランジションは，条件が成立した時に発火（実行）するアプリケーションの機能とする．また，ユーザが対話部品を操作する（ボタンを押す等）ことによって生ずるイベントもこれに割り当てる．

- アーク：プレースとトランジション間の因果関係

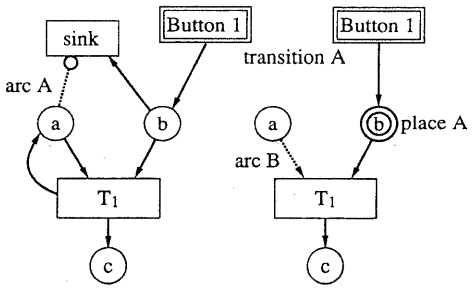
- マーキング：ある時点でのアプリケーションの状態

4.2 ペトリネットの拡張

ペトリネットの複雑化を防ぐために，4.1章で述べた取り決めに加えて，以下のような拡張を行なっている．これらは，いずれもGUIアプリケーションに頻出するパターンであり，この拡張により，ペトリネット作成の負荷が大幅に減少する．

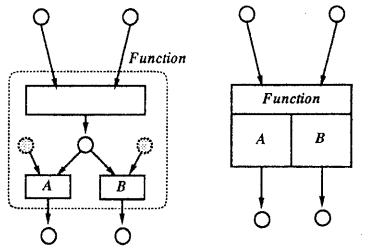
(1) トランジションの発火を抑制する抑止アーク

ボタンを押すとポップアップウィンドウが現れるようなGUIでは，既にウィンドウがポップアップされている場合，そのボタンを無効にする必要がある．また，ポップダウンを行なうためには，ウィンドウがポップアップされていなければならない．このように，GUIでは，1つの状態があるトランジションを発火可能にし，他のトランジションを発火不可能にする，というような場面が数多く存在する．そこで，入力となるプレースにトークンがある場合に発火を禁止する抑止アーク (inhibitor arc) を導入する (図2 a) 中の arc A



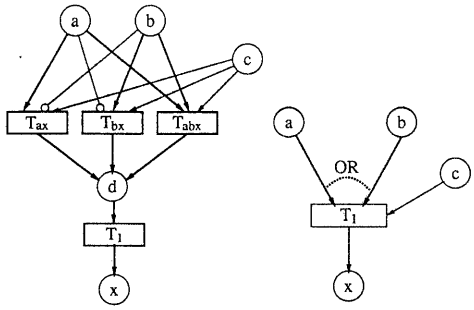
a）抑止アーク　　b）トークン保持アークとトークン消滅プレース

図 2: 特殊なアークとプレース



a) 従来の表現　　b) 拡張した表現

図 3: 複数出力のためのペトリネットの拡張



a) 従来の OR 表現　　b) 拡張した OR 表現

図 4: OR の表現

). トランジションが発火しても抑止アークを通してトークンは移動しない. 抑止アークを用いることにより, ゼロテスト（プレースにトークンがないことのテスト）が可能になる [9][10].

(2) トランジションが発火してもトークンを移動させないアーク

いくつかの GUI 部品は, 固有の状態を保持している. この状態から発火されるトランジションがある場合, 発火にともなう入力プレースのトークンの移動が起こらない方が都合の良い場合がある. たとえば, トランジションが発火してもトグルボタンの状態を保持し続ける場合である. そこで, トランジションが発火してもその入力プレースのトークンを移動させないアークを導入する. 図 2 b) の破線 arc B が, このトークン保持アークであり, 図 2 a) の T_1 からプレース a へのアークが省略できる.

(3) 発火するトランジションがない場合に, トークンを消滅させるプレース

ボタンなどの多くの GUI 部品は, 常にユーザの入力が可能である場合が多い. ところが, 他の条件と伴って動作する部品も少なくない. 例えば, エディタなどのセーブボタンは, 編集中の時に有効になる. このような部品などを自然に表現するために, 発火するトランジションがない場合に, トークンを消滅させるプレースを導入する. 図 2 b) の place A がこのトークン削減プレースであり, 図 2 a) の sink とそれに入る 2 本のアークが省略できる.

(4) 条件による複数の出力を持つトランジション

一般的に手続き呼び出しによる状態の変化は 1 つに決まらない場合が多い. たとえば, ファイルをオープンする手続きでは, その出力は「ファイルがオープンされている」状態と「（何らかの原因で）ファイルのオープンに失敗した」状態のどちらかになる. そのため, 本手法では 図 3に示す例のように, トランジションの出力の分岐を扱えるように拡張する. 起こり得る全ての状態を考慮すればこのような拡張をする必要はないが, ペトリネットが複雑になるのは避けられない. ある程度の抽象化を行ない, 不必要な部分を隠蔽する方が得策である. また, このようなトランジションは, ユーザの選択によって複数の分岐を行なうプルダウンメニューなどの GUI 部品を表現するのに適している.

(5) OR 表現の追加

GUI では, ある機能を実行するための操作が複数存在する場合が多い. ショートカットキーや, ファイル選択における確定方法（OK ボタンとダブルクリックなど）が良い例である. ところが, トランジションの発火条件は, 「入力プレースのすべてにトークンが存在する」という AND 型の条件となっているため, OR 型の条件による動作の記述をするためには工夫が必要である. 図 4 a) に示すネットはプレース a, b のどちらか, または両方にトークンがあり, かつ c にトークンが現れた時に発火可能になる. このように OR 型の条件の記述は複雑なものになってしまう. そこで, 本手法では, 図 4 b) に示すように OR 表現を記述できるように拡張している.

4.3 ペトリネットの段階的詳細化

ペトリネットの作成は, 基本的には, 1 つの機能に注目して, その機能（トランジション）の入力プレース（前提条件）と出力プレース（終了条件）だけを決定すれば良い. しかし, 大規模なシステム全体を 1 つのペトリネットグラフで記述することは非常に難しい. これに対しては, プログラムと同様に, 段階的にペトリネットを詳細化していく

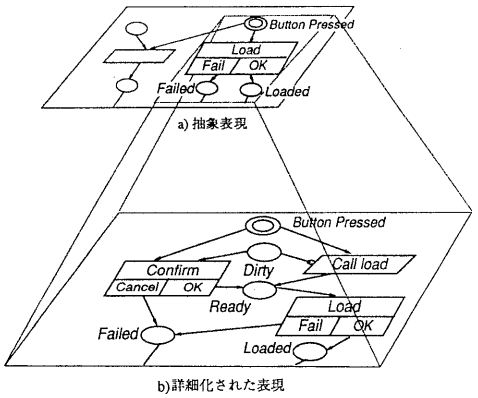


図 5: ペトリネットの階層

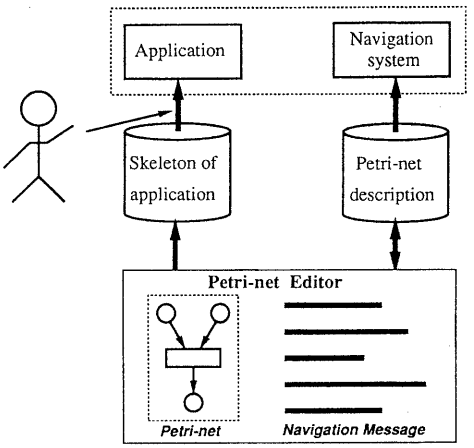


図 6: ペトリネットエディタの生成物

手法が有効である．本手法では，ペトリネットの階層的な定義が可能である．上位のレベルのペトリネットでは，システムの大まかな構造を記述する．次に，各トランジションを，その入力と出力に注意して，さらに抽象度の低いペトリネットへと詳細化していく．したがって，ペトリネットが階層的に定義される時，対応するナビゲーション情報も同様に階層化される．即ち，上位階層においてナビゲーションが要求された場合には，概略的なメッセージが出される．このトランジションが階層的に詳細化されている場合に，そのペトリネットを表示させ，そこで指定を行なえば，より詳細なメッセージが得られる．

ペトリネットの詳細化の例を図 5に示す．図の a) は単純に Load 機能として表現されているが，b) では Confirm 機能を実行するように詳細化されている．ただし，a) における入出力と b) における全体の入出力は一致しなければならない．

4.4 ペトリネットエディタ

図 1 の下側に示すように，ペトリネットを作成するためのグラフィカルなエディタを用意している．開発者はこのエディタを用いてアプリケーションの構造を表すペトリネットおよびナビゲーション情報を作成する．

このエディタは以下の機能を持つ．

- ペトリネットグラフのグラフィカルな編集
- ナビゲーション情報の編集
- 作成したネットのシミュレーション
- アプリケーションプログラムの部分的自動生成
- ナビゲーションシステムが使用するペトリネット記述の生成

システム開発者は，自動生成されたプログラムの骨組みに，アプリケーション本来の機能をプログラミングし，システム全体を完成させる．図 6 にペトリネットエディタの生成物とその流れを示す．アプリケーションプログラムの部分的自動生成については，6.1章において詳述する．

X Window System Ver.11 を用いて作成したペトリネットエディタの表示画面の例を図 7に示す．

5 ナビゲーションシステム

本ナビゲーションシステムが持つペトリネットは，常にアプリケーションの状態を反映している．アプリケーションの状態が変化すると，ペトリネットのマーキングも変化する．ナビゲーションシステムは，ユーザからの要求があると現在のマーキングから最適なナビゲーション情報を選び，それを表示する．

ナビゲーションシステムは，アプリケーションの GUI とは独立なインターフェイスを備えている．このインターフェイスは，機能名を表示するブラウザと，ナビゲーションメッセージを表示するウィンドウからなり，ユーザがブラウザから機能名を指定することにより，ナビゲーション情報を出力する (図 8)．ナビゲーションシステムへの入力は，アプリケーションからの状態の変化と、ユーザからのナビゲーション要求だけである．

図 9に示す例を考える．この例は第 2章で述べた例を含む音声を録音／再生システムの一部である．図 9では，ある曲の再生をする場合に，メニューを開くところからその曲の再生を終了するところまでを記述している．

図 9 に示すような状態でユーザが再生機能のナビゲーションシステムを要求した場合を考える．ナビゲーションシステムは，Play 機能への入力を逆にたどる．こうすることによって，「File Select 機能を用いてファイルを選択する」と，「Play Button を押す」という方法を示すことができる．ここでは，ファイルが選択されていないので，さらに File Select 機能のナビゲーション情報をユーザに表示することができる．このように，ナビゲーションシステムはトークンが現れるまでペトリネットを遡っていく．トークンがあるプレース以前はナビゲーション情報の出力をしない．これにより，最終的にユーザは必要最少限の情報で Play 機能を実行するのに必要な条件を得ることができる．

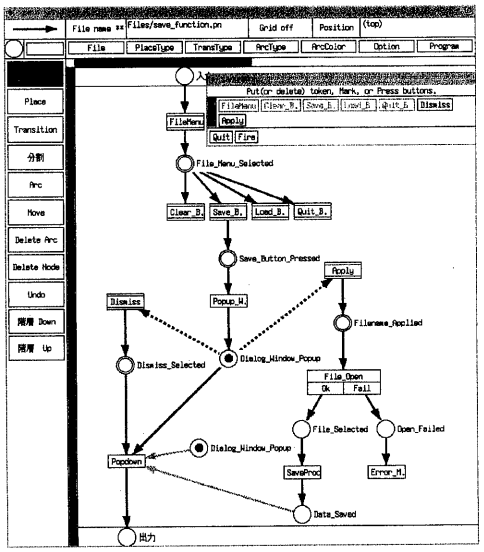


図 7: ペトリネットエディタの表示画面の例

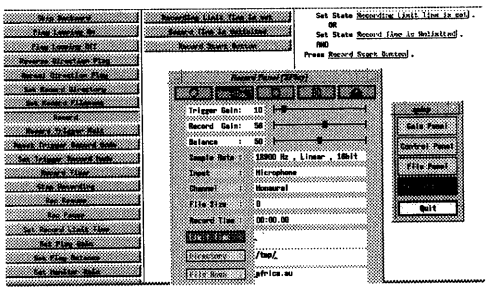


図 8: ナビゲーションシステム

6 アプリケーション開発の支援

6.1 手続きの自動生成

ある手続き（トランジション）に着目して，その出力プレースを調べることにより，その手続きが変更する可能性のある状態を知ることができる．従って，その手続き内に状態変更要求をおこす機構（以降，状態変更手続きと呼ぶ）を自動的に埋め込むことができる．本手法では，機能の実行はアプリケーションプログラム内の手続き（関数）の呼び出しと定義する．

図 10に自動生成される手続きの例を示す．図の a) に示すペトリネットグラフから，b) の手続きが自動生成される．図中の関数呼び出し ChangeState (...) が状態変更手続きである．開発者は，図中の *Manufacturing part* の部分にアプリケーション本来のプログラム（そこから呼ばれる手続きなどを含む）を記述する．*Manufacturing part* から呼ばれる手続きなども，開発者が作成するが，それらの手続きの中に状態変更手続きを記述してはならない．つまり，状態変更手続きは自動生成された部分にのみ存在する．

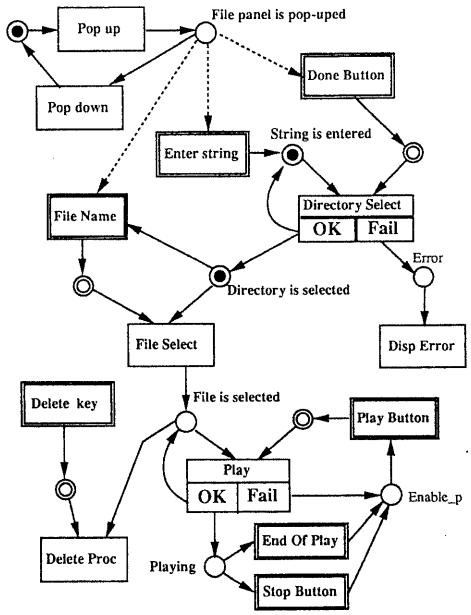


図 9: 再生機能を表すペトリネット

このような規約を確立することにより，トランジションの出力プレースからアプリケーションの骨組みを自動生成することができる．

6.2 処理の制御

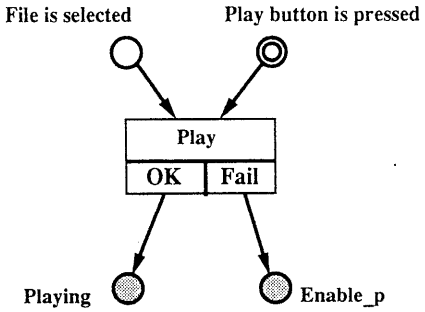
各トランジションを発火させる，つまり，登録されている手続きを呼び出すのは，入力プレースとなる状態が満たされた時である．本システムでは，アプリケーションの処理の流れは状態管理部（ペトリネットエンジン）によって制御される．状態管理部は常に状態を管理し，あるトランジションの入力プレースにトークンが集まった時にそのトランジションを発火（手続きの呼び出し）させる．同様に，呼び出された手続きは出力プレースにトークンを移動させるように，状態管理部に状態の変更要求を出し，次のトランジションの発火へと結び付いていく．状態管理部は，トランジションの入力プレースから if-then 式に自動変換されたものから構成される．実行時はこの if-then 式を繰り返し評価する．

本手法を用いて作成されたアプリケーションの内部構造を図 11に示す．太線で示した部分が自動生成される．図中の Proc は自動生成される手続きで，破線で示される部分をプログラミングしてアプリケーションの肉付けを行なっていく．

7 適用例と評価

7.1 適用例

本システムの評価を行なうために，いくつかの事例に本システムを適用した．なお，本システムおよび事例は，す



a）ペトリネットの例

```
Play ()
{
   int state;
              /* Manufacturing part */

   if (state == OK) {          /* State OK */
     ChangeState (Playing);
   } else {                    /* State Fail */
     ChangeState (Enable_p);
   }
}
```

b）生成される手続き

図 10: ペトリネットと生成される手続き

表 1: ペトリネットの拡張の有効性

事例	プレース (個)	トランジション (個)	アーク (本)
事例 A	74 (17)	57 (12)	142 (11)
事例 B	130 (67)	202 (99)	423 (73)
事例 C	125 (17)	93 (34)	223 (38)
事例 D	63 (19)	46 (11)	117 (18)

べて SUN SPARCstation 10, SunOS 4.1.3 + X11R5 上で実現されている．ナビゲーションシステムのブラウザは C 言語 + X11R5 でおよそ 2340 行，ペトリネットエディタはおよそ 7500 行である．

7.2 開発側の評価

表 1で，事例 A が 5章で紹介したアプリケーションである．事例 B は，事例 A と同様な既存のアプリケーションで，事例 A より多数の機能を備えているものを，本手法に沿って改変したものである．事例 C は，本手法で用いるペトリネットエディタをそれ自身で描き直したものである．事例 D は，アーカイブデータベースのインタフェイスである X11 ブラウザ (Xarchie ツール) を書いてみたものである．なお，各事例は，異なる開発者が作成した．

表 1は，各事例のプレース，トランジションおよびアークの数を示しており，括弧内の数字は，そのうち本手法で拡張されたものの数を示している．この結果から，新しく導入したものが，プレースについて約 30%，トランジションについて約 39%，アークについて約 15% 使用されており，拡張が有効であったことがわかる．しかし，開発者個人によるペトリネットの記述の仕方にばらつきがある．これを解決するために，ペトリネットを記述する上でのガイドラインを作る必要性があると思われる． 表 2は，事例 A，B および C の総行数 (ナビゲーションシステム部分は含まない) と，自動生成された手続き部分および状態管理部の行数を示している．すべて C 言語で実現されており，自動生成された部分は総行数の中に含まれている．既に稼働している事例 B のシステムは約 11000 行である．本システムを用いての再構築においても，その処理内容に依存するのは，約 10450 行 (13000 行 - 1600 行 - 950 行) であり，若干減少している．これは，処理の流れに関する部分を状態管理部に移行したためである．また，この再構築による実行時間の増加はわずかであり，実用上の問題は起きていない．

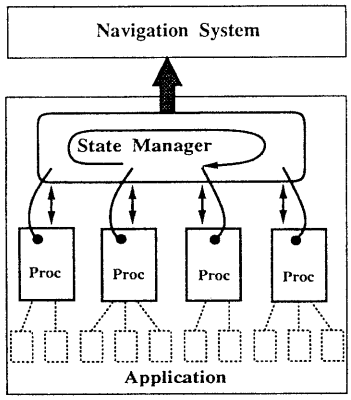


図 11: アプリケーションの構造と自動生成部分

表 2: 試作システムの規模

事例	総行数 (行)	手続き (行)	状態管理部 (行)
事例 A	4500	500	300
事例 B	13000	1600	950
事例 C	8900	1050	720

本手法では，ペトリネットを作成するという付加的なコストを伴う．しかし，それにより必要とするナビゲーション情報を的確に提示できるという大きな利点の他に，要求分析の段階からの支援と，プログラムの品質の向上という利益を得ることができる．

7.3 ユーザ側の評価

7.2 章で示した事例 B を例にとり，その中心的な機能である再生，録音，次曲，および，早送りの操作法を，アプリケーションの状態を変更しつつ表示させてみた．この評価においては表示されるナビゲーション情報の量をメッセージ数で表すことにする． その結果を表 3に示す．なお，表中の数字は，ナビゲーションシステムに表示されるメッセージ数である．初期状態は，アプリケーションに対して，何も操作が行なわれていない状態である．状態 A は，音

表 3: メッセージ数の変化

機能	初期状態	状態 A	状態 B
再生	8	5	1
録音	7	5	3
機能	初期状態	状態 B	状態 C
次曲	9	3	1
早送り	9	3	1

声ファイルがあるディレクトリを指定した状態である．状態 B は，音声ファイルを指定した状態である．状態 C は，再生中の状態である．

初期状態でのメッセージ数は，通常のオンラインヘルプシステムのように，状態によるメッセージの変更を行なわない場合に出力される数に等しい．各状態におけるメッセージ数の平均は，初期状態でのメッセージ数の約 33% で，本手法によるメッセージ数の削減が有効であることがわかる．

7.4 今後の課題

ユーザによる使用実験において，オプショナルな機能に関するナビゲーションが行なわれないという指摘があった．例えば，再生機能の説明で，音量は直接操作には関わってこないが，重要な要素である．このような場合，本システムでは，表示できないという問題点が明確になった．また，初期値などを与える場合にも同様の問題が生じる．例えば，ディレクトリの指定で初期値が起動ディレクトリであるような場合，ディレクトリに関するメッセージは表示されない．何らかの新しい種類のアークを導入するなどして，これらの問題点を解決する必要がある．

本手法では，ペトリネットエディタでシミュレーションを行なうことにより，要求分析の段階からのシステム開発支援を行なうことができる．即ち，十分なクライアントとの対話からシステムを決定し，アプリケーションプログラムを作成する．しかし，要求の変更が起こった場合，本手法ではペトリネットの変更から行なう必要がある．自動生成されるアプリケーションの骨組みが変更される場合には，既に手を加えたプログラムと対応させるのが非常に困難である．要求の変更によるアプリケーションプログラムの変更箇所が最少になるような手法を考案することが望まれる．

本手法では，ユーザが既に行なった意志決定については，その知識を持っているという非常に簡単なユーザの理解モデルを用いている．今後の展望としては，各プレースの過去の履歴を残し，それからさらに有効なナビゲーションを行なうシステムを目指している．

8 おわりに

本研究では，アプリケーションの状態によって制御するナビゲーションシステムのメカニズムの提案と，それを実現するためにペトリネットを用いたナビゲーションシステム作成及びアプリケーションプログラムの部分的自動生成を行なう枠組を示した．各種の評価により，ユーザフレンドリなナビゲーションシステムの開発支援を行なうことができることがわかった．今後は，7.4節で述べた課題に対する解決を試みながら，ユーザへの適応性をさらに上げていくことを考えている．

参考文献

[1] 西田, 高松：知識を用いた説明テキストの理解と情報抽出, 情報処理学会論文誌,Vol.31, No.3, pp.481-490 (1990)

[2] M.Green : A Survey of Three Dialog Models. ACM TOG Vol.5, No.3, (1986).

[3] P.D.Wellner. Statemaster : A uims based on statecharts for prototyping and target implementation. In CHI'89, pp. 177-182. ACM, (1989).

[4] C.Janssen, A.Weisbecker, J.Ziegler : Generating user interface from data models and dialogue net specifications. In INTERCHI '93, pp. 418-423. ACM, (1993).

[5] Willem R. van Biljon : Extending Petri nets for specifying man- machine dialogues. Int. J. Man-Machine Studies 28, pp.437-455, (1988).

[6] Working Party X/3. Draft Recommendation Z.120-Message Sequence Chart(MSC). CCITT, (1992).

[7] M. Ajmone Marsan et al.: TOPNET: A Tool Based on Petri Nets for the Simulation of Communication Networks, Tool Description, PNPM '91, also in IEEE Journal on SAC, Vol.8, No.9, pp. 1735-1747(1990).

[8] 孫, 落水：ペトリネットによる並行ソフトウェアシステムの設計時動作解析, 情報処理学会論文誌, Vol.29, No.8, pp.782-789 (1988)

[9] K. Jensen and G. Rosenberg (eds.): High-Level Petri Nets, Theory and Application, Springer-Verlag, Berlin (1991).

[10] W. ライシッヒ著, 長谷川, 高橋訳：ペトリネット理論入門, シュプリンガー・フェアラーク東京 (1988)