ソフトウェア工学 86－T 5
（1992. 7. 21）

ネット指向ソフトウェア構築ツールについて *

— On Net-oriented Software Tools —

寺内 睦博 †
Mutsuhiro Terauchi

広島大学 工学部 第2類
Department of Circuits and Systems, Hiroshima University

1 はじめに

ソフトウェア工学ではその生産効率性および理論的な立場から開発工程をサポートする環境に関する研究が行われ，数学的な基盤を持つ概念が展開されつつある．一方，ネット理論の研究においては，理論的探究の成果を応用面に活かしたいという強い欲求があるようである．近年，両者の研究領域は相互の分野へ活路を見いだし，それぞれの成果を参考に融合した研究が行われている [1].

本稿では，ペトリネットユーザという立場から，最近提案されたいくつかのネットワークツールの概要を紹介すると共に，Design/CPN というひとつのツールを取り上げ，その構成と使い心地に関する感想およびそのソフトウェア生産への適用可能性について述べる．

2 CASE について

ソフトウェア生産の現場では，近年の対象とするシステムの大規模化，および複雑化に対して様々な努力がなされてきている．このうち，ソウフトェアの生産効率，品質向上を目的として，計算機上の各種の支援ツールを用いてこれらの改善を行うアプローチがある．これは CASE と呼ばれ，コーディングの支援からシステム全体の仕様記述のサポートまで各レベルに応じた環境が提案されている．

本章では，これらの概要についてツール群に求められる望ましい条件という観点から簡単にまとめてみることにする．

2.1 構造化手法

構造化プログラミングやソフトウェア工学技法が，主にプログラム構造，開発工程，開発支援ツールに強い影響を与えているが，これらコードレベル，モジュールレベル，システムレベルの3つに共通するのは構造分析概念である．[2] システム設計はまず仕様設計を行い，それらをしだいに詳細化して実際のコーディングが行われる．その方法論として，機能分解法，データフロー設計法，データ構造設計法がよく利用される．

ソフトウェア工学技法の代表的な例を以下に示す．[2]

Structure
- Top Down Design
- Virtual Machines
- Hierarchical Modular Programming
- Information Hiding
- Data Abstraction
- Bottom Up Design
- Structured Code

Process
- Teams
- Walkthrus
- Reviews
- Top Down Implementation
- Continuous Integration
- Phased Builds

Tool
- High Level Language
- Development Support Library
- Programmers Work Bench
- Program Librarians
- Automatic Test Generation
- Cause-Effect Charts

プログラム構造を取り扱っている技法は，ツールやプロセスに関するものの基盤となっている．もちろん支援ツールや開発工程はプログラム構造に強く影響を及ぼすが，必要なプログラム構造に適応する技法を選択する必要がある．

2.2 JSD 法

JSD 法はマイケルジャクソンの提案したシステム開発手法であり，従来の構造的手法におけるウォーターフォールモデルによるソフトウェア開発の問題点を考慮した上で，異なる視点からそれを解決した開発手法である [3]．この手法では，モデルを記述するためにデータフロー図に類似したシステム仕様図と，逐次型プロセスの非決定的事象遷移図であるジャクソン構造図を用いる．このように JSD 法では状態遷移を考える代わりに，それの因子である事象に着目し，状態を変化させる事象（イベント）の遷移関係をグラフ表現し，対象の構造としている．

この手法では，まずモデル化する対象の要素を特定する．その要素に関して可能な動作を考慮し，それらの組み合わせから属性や変数を決定する．これらの動作を時間的な推移を考慮してグラフ化する．その後，これらの相互作用を表現し初期モデルが作成される．さらにイベントの同期などのタイミングを考慮した後，システムの実現段階に入る．ここでは実システムのファクタを取り込みさまざまな検証の後システムが完成する．

この手法は複雑で非決定的な実システムのモデル化可能な巧妙な方法論であるといえる．

TEL: 0824-22-7111 Ext. 3441 FAX: 0824-22-7195
E-mail: mucha@infonets.hiroshima-u.ac.jp

2.3 上流 CASE ツール

実際の開発現場では不幸にも積極的な CASE 環境の取り込みがなされているとは言い難い [4]. 前節までのシステム開発法の発展にもかかわらず, 低レベルのツール群の利用はあっても, システム全体の仕様記述を行う上流 CASE ツールを導入し, うまく活用しているところはまだ少数派である. 現在の上流 CASE の基本概念は構造化技法であるが, このようなトップダウンな思考法は日本人には受け入れ難いらしい [4]. 今までのプロセス中心のものの考え方からデータフロー図や実体関連図などのオブジェクト中心の発想法への切替が必要である. このためには, モデル表現の違いを即座に転換できる多角的な視点を持てることが要求される.

3 ネット指向ソフトウェア構築ツール

前章までの議論から考えると, 今後の CASE ツールではトップダウン設計技法をサポートし, データの流れやモジュールの関連を図式的に表現でき, さらに数学的な検証方法が確立されているモデルを持つものが必要とされていることがわかる. 本章では, 最近発表されたネットベースツールの概要について紹介する.

1) HOOD Nets: [5] Hierarchical Object Oriented Design Nets (R.Di Giovanni [Pisa])

HOOD (階層オブジェクト指向設計) とはソフトウェアライフサイクルのアーキテクチャ設計フェーズのための標準 ESA(Europe Space Agency) 手法である. これは Ada プログラム開発のためのツールで, 抽象マシンとオブジェクト指向設計概念を結合させたものである. HOOD オブジェクトはデータ構造とオペレーションを内包する private body と, 他のオブジェクトがリソースを利用する際のオペレーションの宣言を含む public interface からなる. このオブジェクトに組み込まれる情報の多くは Ada あるいは疑似 Ada コードで記述されている. これによりソフトウェアの設計とコーディングという 2 つの異なるフェーズでの一貫性の保持が簡単に行える. HOOD ではコードの再利用を薦めているが, 疑似コードの場合, オブジェクトの特性の形式的検証はできない. そのため, FST(形式的仕様技法) を導入する. この形式的技法としてペトリネットのサブクラスであるステートマシンが採用されている. この利点として,

1. オブジェクトの動的ふるまいを形式的に記述できる (形式的検証が可能)
2. HOOD の構造化原理によりペトリネット記述にモジュラリティを付加できる

つまり明示的にオブジェクト指向概念を表現できるということである.

オブジェクトのふるまいは OBCS と OPCS の 2 つのクラスで記述される. OBCS (オブジェクト制御構造) は各能動オブジェクトに結合し, その制御を記述する. OPCS (オペレーション制御構造) は各オペレーションに結合し, 実現アルゴリズムを記述する. HOOD システムは木構造であり, root object という root を持つ. この木は設計プロセス木と呼ばれる.

HOOD Nets の特徴は, オブジェクト制御構造ネットとしてステートマシンを導入していることである. この場合, システムが複雑になるとネットが巨大になるという欠点があるが, ステートマシンを一般あるいはハイレベルペトリネットへ変換することによりコンパクト化し, 見通しをよくすることができる. この HOOD Nets は EUREKA IRENA プロジェクトにて開発中である. [1990 6 月現在]

2) Great SPN 1.5 [6] GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets (トリノ大学 G.Chiolla)

Great SPN はファイル共有によりペトリネットモデルの生成, 解析する多くのプログラムから構成されている. このツールの代表的な機能を以下に示す.

グラフィックインターフェース: Sun View あるいは X View の上で動作し, グラフィックモデルの編集, 階層化をはじめ, 各種の図形的操作, 印刷を行う. また構造的特徴の図式表示, ペトリネットのトークンゲーム, サブモデルの付加, モデルデータベースへのアクセス, タイミングや確率変数の仕様の定義, メニューを用いたネット解析, 実行結果の図式表示, 時間および確率的ペトリネットのシミュレーションなどの様々な処理が行える.

構造的特徴: P インバリアント, T インバリアント, デッドロック, トラップ, 相互排他, 構造的競合, 構造的バウンドなど多くのペトリネットの構造的な特徴を解析できる.

可到達グラフ: 可到達グラフの生成および解析を行うプログラムが提供される. これにより, ホームスペース, デッドロック, ライブロック, トランジッションライブネスの解析を行うことができる.

マルコフソルバ: ある瞬時の性能解析と同様に, 定常状態の解析が行える.

シミュレータ: シミュレーションモジュールによりインタラクティブなイベント駆動シミュレーションが行える. 時間的に双方向のペトリネットシミュレーションと共にその性能評価をリアルタイムに表示し, また任意のイベントの再スケジューリングや周期的サンプリングを伴うバッチ的シミュレーション機能をも提供する.

3) Topnet [7] a TOol based on Petri NETs for the Simulation of Communication Networks (トリノ工芸大学 M.Ajmone Marsan, et al.)

Topnet は通信ネットワークとそのプロトコルのシミュレーションモデルの開発および実行を支援する DEC VMS 上のソフトウェアパッケージである. このツールの意図するものは, 最大限可能なシミュレーション実験の実現とモデル記述の再利用にある. これによりライブラリから持ち出した多くのサブモデルを結合することにより大規模なシステムを構築することが可能となる. Topnet ではペトリネットに時間とデータ操作機能を付加した PROT nets (Process Translatable Petri Nets) を内部記述として採用している. ネットのグラフィック表現と共に, スクリプトと呼ばれる Ada コードを内包し, トランジション発火時にそのアクションが実行される. Topnet のグラフィックインターフェースの特徴として, オブジェクト指向ア

プローチに準じたトップダウン開発に適した階層的グラフィックエディタがあげられる．Topnet は Ada コードに変換可能であり，これを図式的に実行させることもできる．実行時のアニメーション機能と同時に，測定された性能を表示可能である．

4) Voltaire [8] a Discrete Event Simulator (マッギル大学 P.Parent, O.Tamir [現　Bell Canada])

Voltaire はシミュレータと階層ペトリネット記述言語からなる時間付きペトリネットをベースとした離散事象シミュレータである．Voltaire は巨大で複雑なシステムを比較的速く効率的にモデル化し，シミュレートする機能を提供する．このシステムの特徴は複雑なペトリネットのネット要素をブロックとして構造化し，それ自身を階層化し，高レベルブロックとして組織化できる点である．これによりトップダウンモデル化技法をデザイナに提供できる．ペトリネットを導入することにより広範なシステムクラスをモデル化でき，さらに同期・並行事象を簡単にモデル化できるのみならず，それらの依存関係を求めることもできる．トランジションのアクションの記述は C++ 言語で書かれ，トークンには任意の属性が設定でき，大規模システムの特徴を集中管理する大域メモリとリソース制御を行うセマフォを有する．このツールの特徴であるブロック化はオブジェクト指向でのカプセル化原理に準じ，アクション記述言語の C++ とも相まってトップダウン思考の方法論を支援する．もちろんボトムアップ手法による開発も可能ではある．

5) A Software Environment for the Generator, Representation and Analysis of PN basedModels of Manufacturing Systems [9] (ミラノ大学 F.Archetti, et al.)

このソフトウェア環境はペトリネット表現を通してマニュファクチャリングシステム (以下 MS と略記) をモデル化し，解析することを目的としている．この一番の特徴は，MS のグラフィック記述からペトリネットモデルを自動的に生成し，表示することである．これにより，初心者でも簡単に使いこなせることができ，その応用範囲は広がる．以下にこの環境を構成する主なモジュールを示す．

ペトリネットモデルの自動生成： 要素集合が与えられ，それがグラフィックに与えられると対応するペトリネットモデルが生成される．このモジュールにより各要素にデータを連結することができる．

ペトリネットモデルの自動表現： このモジュールではペトリネットが図式的に生成され，表示される．枝の交差を最小にするためにグラフ理論を用いて平面性の判定が行われる．ここでは，論理的に結合したプレースとトランジションは可能な限り近傍に配置するという方策がとられている．

調整エディタ： このモジュールは通常のグラフィックエディタと同等の機能を提供する．このモデルでは，プレースは名前，最大・最小のマーキング，初期マーキングの属性を持ち，トランジションは名前，発火時間，分布 (非決定的な場合) の属性を持つ．さらに競合解消戦略，データ依存型の発火時間を持つトランジションもサポートしている．

解析とシミュレーション： このモジュールはネットの決定的なふるまいのシミュレーションを行う．次の 3 つのイベントにより動作を停止することができる．1) サイクルの長さ，2) トランジションの発火回数，3) プレースのトークン数　またシミュレーション中の各プレースのトークンの分布とトークンの平均数が結果として出力される．これらの結果はそのまま各応用分野の変数として解釈できる．

このソフトウェア環境は C と C++ でインプリメントされている．

6) FORSEE [10] a collection of tools for FORmal System Engineering Environment (テレコムオーストラリア J.Billington, et al.)

FORSEE 環境は現在，TORAS: 可到達性解析ツール，PROMPT: 自動インプリメンテーションツール，Design/CPN: グラフィカルエディタ・シミュレータの 3 つの要素から構成されている．FORSEE はソフトェアの生産性と品質を高めるため，システムの数学的モデルである形式的技法を持つべく開発されたソフトウェア構築環境である．応用分野として意図された電気通信ネットワークとそのサービスは分散し，並行に操作される．多局結合，回線多重化，異種機能の同時利用などのトレンドに電気通信サービスの並行性の量的な増加を予感させる．それゆえ，並行システムを記述し，解析できる数学的手法は必要不可欠である．FORSEE とは仕様を生成・維持し，サービス仕様を形式的に洗練し，シミュレーションを行い，その性能を検証・評価し，テストシナリオを生成する環境である．現在，数学的技法としてペトリネットを採用している．これはペトリネットが並行性を含んだ形で定式化されており，実行可能であり，多くの解析手法を備えた数学的な基盤を持っているからである．さらに図式的でその動作も確認しやすいということもある．FORSEE において，TORAS は検証ツールとして，PROMPT は形式的仕様を C 言語へ変換するツールとして，グラフィカルエディタ・シミュレータである Design/CPN を FORSEE のフロントエンドとして位置付けている．以下にこれら 3 つのツールの概要を述べる．なお，Design/CPN については次章でもう少し詳しく述べる．

TORAS： TORUS はペトリネットの可到達性の解析を行う．これは初期状態から到達可能なすべての状態と状態遷移を生成する．この際，状態数の爆発が起きるが，いくつかのアルゴリズムを利用して，デッドロックやライブロックなどの特性を失うことなく状態空間を小さく抑えている．一例を挙げると，100 のプロセスを持つ簡単な事例ではすべての状態空間は 10^{47} であるが，簡約状態空間は 30,000 であり，デッドロックも保存されていた．また IEEE 802.6 Configuration Control protocol の検証に適用したところ，すぐにデッドロックを検出できた．

PROMPT： PROMPT は通信ソフトウェアの開発環境を提供する．これにはコンパイラ，デバッガ，制御インタフェース，ログファイルアナライザという 4 つの要素がある．コンパイラは拡張ネット言語 (XNL) を入力とし，C 言語コードへ変換する．デバッガは XNL レベルで動作する．制御インタフェースはいくつかの XNL 仕様を同時実行させ，ユーザの事象制御のログを取る．

Design/CPN：Design/CPN はカラーペトリネットの編集とシミュレーションを行うパッケージソフトウェアである．これらには多くの解析手法が存在する．このツールは 同じ代数的基盤の範囲内での抽象データ型とペトリネットとを組み合わせたものであるといえる．

7) UltraSAN [11] Stochastic Activity Networks (アリゾナ大学 J.Couvillion et al.)

UltraSAN は Unix ワークステーションのXウィンドウ上で動作し，各種エディタ，シミュレータ，ソルバから構成される．

モデル化の枠組み: Stochastic Activity Networks は確率的ペトリネットである．これはアクティビティ，プレース，入出力ゲートから構成され，アクティビティには時間付きのものと即時発火の2種類がある．

SAN Editor: ペトリネットのグラフィックエディタであり，Xウィンドウ上で動作する．メニューとマウスを用いたユーザインタフェースを提供する．

Composed Model Editor: SAN Editor で作成した構成モデル(subnet)の構造を定義するエディタであり，グラフィックユーザインターフェースを持つ．階層性あるいは subnet 間の関係を定義することができる．

Performability Variable Editor: ネット上の変数を定義するエディタである．reward, activity という2種類の変数が用意されている．reward 変数はネットの確率に関する変数であり，シミュレーション結果を代入することができる．activity 変数は activity の実行時間をシミュレータより与える．

Reduced Base Model Constructor: ネットの解析を理論的な方法で行う場合，そのシステムの内部表現は reduced base model constructor へと渡される．そこでSAN のsubmodel と実行変数からモデルの定常表現が構築される．この表現には状態集合，状態間の推移率，および各状態の impulse および rate という reward 変数が含まれる．これらの変数に加えて木構造の状態の集合をデータ構造として想定する．これを初期状態として，subnet およびそれらの関係を用いて状態を遷移させ，これにより各パラメータが求まる．これらの処理は，AnalyticalSolver (Direct Steady State Solver, Interactive Steady State Solver, Transient Solver) によって行われる．DSSS は LU 分解を基本とし，一般化 Markowitz 戦略および Osterby and Zlatev の手法により定常変数の各解を求める．ISSS は繰り返し緩和法に基づく．これは状態空間が巨大になる場合に適用されるが，収束の保証はない手法である．収束化因子の決定はユーザが試行錯誤で行う必要がある．TS は乱数を用いてある時刻の変数に対する解を求める．この方法はマルコフ過程をポアソン過程に従属させるというアイデアを用いている．この方法は空間的にも時間的にも効率はよい．

Simulation Solver (Simulator): ネットの解析を実験的に行う場合，そのネットは SImulator へと渡され実際に実行される．このシミュレーションは前述の木構造の状態表現の上で実行される．この状態遷移では木上のノードが submodel に相当するので，状態の数を抑えることができる．このため，activity 集合の中で代表的な activity を求める．

このツールはランダム化手法の拡張とXウィンドウ上で動作するという簡便さを特徴としている．

4 Design/CPN

本章では，Meta Software 社の Design/CPN という開発ツールの概要について述べる．以下にこのソフトウェア・ツールの簡単な紹介をする．

最近の20年間でペトリネットの研究は複雑な並行システムの挙動特性の研究と関連して理論的な発展を遂げてきた．同じ時期にペトリネットは通信プロトコル，電話網のスイッチング・ネットワーク，分散データベース，FMS，ソフトウェア・デザイン，ハードウェア・アーキテクチャの各分野における巨大システムの仕様記述や検証に実用化されるようになった．そのほかにも OA, FA およびマン・マシンインタフェースでもペトリネットを利用する研究が行われている．このようにペトリネットは新規システムの設計，現行のシステムの仕様記述やその検証，迅速なプロトタイプ作成などのシステム開発プロセスにおいて従来とは異なる立場で利用することができる．

ペトリネットを実システムに応用する際に大きな障害となるものの一つにネットの大きさの問題があげられる．これは対象とするシステムが小さなものであってもこれに相当するペトリネットを記述すると途端にネットの規模が大きくなるという問題である．このモデルサイズの大きさの問題はペトリネットに限らず，全ての種類のシステムモデルで遭遇するものである．この問題は次に挙げる事柄の実現によって解決されると考えられる．

1. 1970年代後半のハイレベルネットの理論的研究
2. 1980年代中盤の廉価なグラフィックワークステーションの登場
3. 1980年代後半の大規模階層型グラフィックモデルをサポートするソフトウェアの開発

Colored Petri Nets の創始者である Aarhus 大学の Kurt Jensen ら，および Meta Software 社により階層色付きペトリネットをサポートするソフトウェアパッケージ Design/CPN が製作された．

さて，システムのモデル化と設計の分野では単一レベルのシステムモデルでは，見晴らしの悪さ，詳細記述の繁雑さ等，不十分であることが広く知られている．これらの問題を克服するために階層的モデル化言語が導入され，実際の現場でよく使われるようになった．そのほかにも SADT や IDEF がある．そのような階層的モデル化言語は，(1) 詳細な記述の隠蔽，(2) モジュール分割，(3) ソフトウェアの再使用可能性，(4)top down および bottom up の両方の開発戦略，(5) 強力なグラフィック表現能力等の長所を持つ．さらに，モデル化言語は次のようなことがらを満足しなければならない．

- 正確で一貫性のある要素に対する行動概念のサポート
- 異なったレベルにおける巨大で複雑なシステムモデルの実行を観測可能なこと

上述のグループのモデル化言語の多くはこれらの実行可能性という性能を持っていない．これに対して，最近のハイレベル・ペトリネットは自然で堅実に並行動作を表現する秀れたモデル

化言語として受け入れられつつある．その理由の一つはハイレベル・ペトリネットの数学的な基盤にある． [15] - [19]

4.1 色付きペトリネット

今日，ペトリネットの最も現実的な応用はハイレベル・ペトリネットを用いたものである．ハイレベル・ペトリネットと普通のペトリネットとの関係は高級プログラミング言語とアセンブリ言語との関係とほとんど同じと言ってよい．つまりハイレベルは巨大システムの複雑性を管理する能力をユーザに持たせつつ，より高い構造化能力を有する．色付きペトリネットと述語トランジションネットは有力な種類のハイレベル・ペトリネットである．それらは特にシミュレーションに関しては密接に関連している．

色付きペトリネットは複雑なシステムを簡潔に表現するために，数学的表現能力とグラフィックモデルの明快さを合わせ持つ．ある任意の色付きペトリネットの挙動は普通のペトリネットと等価であるが，このとき後者は述語は持たないものの，非常に多くのグラフィックオブジェクトを持つことになる．色付きペトリネットの定型的述語はカラーセット，マーキング，アーク表現，ガードからなる．色付きペトリネットの簡単な紹介は文献 [15], [20] に，詳細およびその数式的な定義については文献 [16] を参照のこと．

Design/CPN には階層色付きペトリネットのインタラクティブな編集をするためのエディタが付属する．つまり色付きペトリネットは一つのサブネット（もちろんこれがさらに下位のサブネットを含んでもよい）を含んでもよい．またこれらは形式的に相互作用する．このサブネットはページと呼ばれ，代入，呼出，融合という3種類の異なる関係を持つ．これらの関係は有効な構造化手法を構成し，モジュール的で管理の容易な大規模な色付きペトリネットを構築することができる．シミュレーション，可達グラフ，プレース・インバリアントの三種類の CPN 解析手法の全てが拡張可能なので，それらは階層的な CPN へも適用可能である． [21], [22] Design/CPN は次の事柄をモデル化し，解析するために使われ続けている．

- 命令制御センタ：IDEF/CPN と CPN は巨大な命令，制御，通信，インテリジェントシステムにおける人間の能力を解析するために用いられた．
- 新しいハードウェア・チップ：簡便な論理シミュレータの代りとして，チップ・デザインの階層的グラフィック記述をモデル化し，解析するために CPN が用いられた．
- 複雑な貯蓄操作問題：IDEF/CPN と CPN は大きな資産を扱う都市銀行の資産流通部門におけるトランザクションの流れに対するリアルタイムモデルの解析と検証のために用いられた．

その他，各分野への潜在的適用可能性があると考えられる．

4.2 Standard ML

[23] - [25] Standard ML (SML) は Robin Milner をリーダーとする設計グループによってエジンバラ大学の計算機科学研究所で開発された関数型プログラミング言語である． Design/CPN の中の ML の評価エンジン (CPN/ML) はこの SML コンパイラ (ESML) を基盤としている．

言語として Standard ML の採択した理由は，それがデータ型を持つ関数言語で，かつインタプリタであることに基づいている．Standard ML 次のような理由でこれらの要求に適合していると言える．

- 関数型言語
- CP グラフから CP 行列への変換の容易さ
- インタプリタ
- 型制約の強い言語
- 抽象データ型のサポート
- 多相型システム
- オーバローディング可能
- 静的なスコープによる透視性の向上
- 例外処理機構
- モジュラリティ

4.3 IDEF

Design/CPN には IDEF の図を作成し，それらの図を同等の CP-net に変換するプリプロセッサ機能がある．IDEF は階層的システム記述方法として広く使われており，SADT という名前でも知られている [12], [28].

IDEF/CPN は IDEF の一つの方言である．これを用いてユーザは，実行可能な挙動モデルを定義することができる．これと IDEF との違いは枝表現に対するさらに精密な文法を設定していることである．これにより，IDEF のアクティビティと CP-net のトランジション，IDEF の枝と CP-net のプレース，IDEF の枝表現と CP-net のアーク表現，および IDEF の階層と CP-net の階層の間には，ほとんど1対1の対応関係がある．

IDEF/CPN のユーザは CP-net を用いて解析を行うであろうが，CP-net は元の IDEF/CP N と同じ基本レイアウトを持つので，この差はほとんど分らない．

4.4 Design/CPN の実行

Design/CPN では二つのモードが存在し，それを使い分けることができる．しかし，これらは同一プログラムであるため，データの受渡しなどは不要である．

4.4.1 編集モード

Design/CPN を使うことによりユーザは複雑な階層的色付きペトリネットを構築し，編集することができる [29]．高解像度スクリーンとマウスを装備したグラフィックワークステーション上で色付きペトリネットのグラフを直接操作することができる．全ての編集操作は即座にスクリーン上に反映され，現在の CP-net は常に正確に画像上に示される．

Design/CPN のエディタは CP-net に含まれる異なる種類のオブジェクト（プレース，トランジション，アーク，初期マーキング，ガード，アーク表現等）の各々についてデフォルト属性（サイズ，形，濃淡，フォント，色等）を持っている．ユーザはこれらの全てのデフォルトを変更することができる．また

Design/CPN のエディタは非常に多くの文法チェック機能を持っている．これらの機能はプログラミング言語での型チェックに相当する．つまりシステムにより潜在的に安全でない構造が検出された場合，それをユーザに教えてくれるのである．

ユーザは新しいページを生成することによりトップダウン階層を構築でき，現存するサブネットを新しいページに移動することによりボトムアップ階層を構築できる．さらに，CP-net のレイアウトには大きな自由度がある．

4.4.2 シミュレーションモード

Design/CPN のシミュレーションでは，階層的 CP-net の各々のページはそれ自身のウィンドウに表示される [27]．発火可能なトランジションはシステムによりハイライト表示され，シミュレーションステップの発火がアニメートされる．これによりユーザはトランジションへ向けて入力枝をトークンが移動したり，出力トークンを見ることができる．Design/CPN のシミュレータは多数のオプションを持っており，それにより色々な方法でシミュレーションプロセスを実行・観察できる．

シミュレーションはその各ステップをユーザが定義するという意味ではマニュアルに実行可能である．またその各ステップがシステムにより生成される（必要ならば，ランダムな選択を用いて）という意味では自動的に実行することもできる．いずれにせよ，シミュレーションステップの影響を計算するのは Design/CPN のエディタである．

マニュアルモードでは，ユーザはシミュレーションステップ全体にわたる完全な制御が可能である．しかし，この場合でもシステムから多くの手助けを得ることができる．例えば，ユーザは Design/CPN に対して，あるトランジションについてバインドし得る全ての値を計算させることができる．このようなシステムの機能により，ユーザはステップ計算に労力を費やすことなしに，シミュレーションの理解に集中することができる．

4.5 Design/CPN Palette によるネット解析

Design/CPN はユーザに階層的 CP-net の構築とシミュレーションを許容する．シミュレーションは，システム設計の挙動特性を解析したり，プロトタイプ作成やインプリメンテーション，および性能評価の支援に使用可能な有効なツールである．しかしながら，あるシステムが論理的に正常に動作するかどうか，つまり仕様に適合しているかを検証するためには，さらに形式的な解析方法を採用しなくてはならない場合が多い．CP-nets を使用する重要な根拠は，モデル化されたシステムの特性が形式的に証明することができる多くの形式的解析方法を提供できる点にある．

Kurt Jensen および Design/CPN を開発した研究チームで構成される Colored Petri N et 開発プロジェクトは将来，形式的な解析をサポートするように Design/CPN を拡張する予定である．これはモデル化されたシステムの特性を数学的に検証できるような開発環境を提供する．この拡張バージョンは Design/CPN Palette としてリリースされる予定である．

形式的な解析方法には 4 つのクラスのものが存在する．1) 可達グラフの構築（到達可能な全てのマーキングを表現）2) 線形システムインバリアントの計算と解釈（プレースインバリアントおよびトランジションインバリアント）3) 構造的特性の検証（挙動特性）4) リダクション（ある選択された特性の集合を変化させずにネットを縮約する）　ここでは可達グラフ解析と P インバリアント解析のサポートについて説明する．

4.5.1 可到達グラフ

Design/CPN Palette では，ユーザが CP-nets に対する可到達グラフを構築したり，解析したりすることができる．この構築と解析はともにほとんど自動的に行われる．つまり，ユーザからは些細な命令により，それらはシステムによって実行されるのである．

Design/CPN Palette は可達グラフの構築に対して新しい手法を提供する．この方法では，ユーザは各々のカラーセットに対して可能な順列の集合を定義することができる．順列集合には，その順列が代数群を形成し，この制約が CP-net のマーキングにおいてその順列が同等な関係を誘導するという十分な保証が得られなくてはならないという要求が存在する．通常の可達グラフは CP-net の到達可能なマーキングの各々に一つのノードを割当ててある．同クラスの可達グラフ (OE-graphs) は各々の到達可能な同等のクラスに対して一つのノードしか割当てていない．OE-Graph は各々，ノードとアークの名前集合を一つ持っており，これを使って対応する可到達グラフを再構成することができる．

各々の OE-graph には，対応する可達グラフと同じ情報が正確に保持されているので，可到達グラフと同様な方法で，デッドロック，マーキングの最大バウンド，基本状態，ライブネス，到達可能性などのシステム特性を検証することができる．これらのシステムの諸特性は OE-graph を用いて全てのバウンドされた CP-nets に対して決定可能である．さらに，その解析は自動的に実行可能である．つまり OE-graph は Design/CPN Palette により構築でき，その後，CPN/Palette により自動的に解析される．ユーザの行うべきことは，ただ許容される順列群を定義するだけである．

Design/CPN Palette における OE-graph の自動解析は標準的なグラフアルゴリズムを提供する．上述の全ての特徴は，強連結成分 (SCC graph) の計算と単純な探索（例えば，最大の係数の検出など）で実現される．

自動解析法に加えて，ユーザは，無視するプレースやトランジションを指定することにより OE-graph を縮約することができる．例えば，ユーザが通信プロトコルの解析中に再転送を無視して，送信元から受取り先までの順方向のトランジションだけを考慮することができる．

OE-graph は通常の可到達グラフと同じ演繹能力を持っているが，そのサイズは非常に小さい．典型的な例として，ペトリネットの文献でよく使われる例である分散データベース・システムを考えてみよう．そのデータベース・システムは n 個の異なるサイトで構成されているとする．この時，可達グラフの大きさは（ノード数）は n*3n のオーダーで増加する．一方，対応する OE-graph の大きさは n2 のオーダーでしか増加しない．

4.5.2 P インバリアント

Design/CPN Palette では，ユーザは構築された CP-nets に対する p インバリアントを見つけることができ，それらの p インバリアントは，プログラムの検証のときのインバリアントの利用方法と同様に，CP-nets の特性の解明に用いることができる．CP-net の p インバリアントは個々のプレースに付加された重みの集合である．Design/CPN Pale tte の中では，p インバリアントを次のように見つけることができる．

リダクションプログラムを使って，ユーザはインバリアント集合が変化しないように C P 行列を小さくすることができる．これはリダクションルールのいくつかを使ってできる．このリダクションはインタラクティブ・モードで実行できる．そこではユーザがリダクションルールを定義でき，システムはその正当性をチェックし，その影響を計算する．しかし，リダクションは通常，もう少し自動的なモードで行われる．このモードでは，リダクション方策は CPN ML で書かれた小さなプログラムで定義される．このような ML の使い方は興味深いものである．というのはそれはシステムを検証する LCF 理論のための制御言語として開発されたという ML の元々の目的に逆戻りするからである．また上級ユーザは CPN ML でプログラムを書くことにより，独自のリダクション方策を定義して用いてもよい．しかしほとんどのユーザはデフォルトの標準リアクション方策を使用することになる．このような戦略を使うことは，上述のマニュアルのリダクション・モードに比べ，スピードの点で驚くべき利点となる．

二番目のプログラムは p インバリアントを見つける手助けとなる．このためには，ユーザは CP-net グラフを直接扱うことになる．ユーザがいくつかのプレースの重みを定義すると，システムはこれらの重みの意味するところ，つまりユーザが定義した重みから決定される重みを計算してくれる．この計算ではシステムは縮約された CP 行列を利用するが，結果として出力される重みは元の CP-graph についてのものである．ユーザは計算された重みを調べて，もう少し重みを付け足して重みを計算し直すかを決定する．この方法はスプレッド・シートの操作に似ている．CP-net はスプレッド・シート・モデルを定義する．つまり，それはユーザが定義した重みを用いて未知の重みを計算するルールを定義することになる．

その方法は原始的で厄介であるように思われるが，そうでもないのである．逆に，いくつかの重みを決めることにより，興味深いインバリアントを見つけることも結構多いのである．何故こういうことができるかを理解するためには，CP-net がスパースで，ネットの異なる部分間では強い相互作用を持つことを思い出さなくてはならない．これらの特性によりシステムの中のいくつかの重みを与えることにより，多くの重みを計算することができる．さらに，インバリアントは始めから見つかることはめったにないのである．しばしばユーザはモデルがうまく動くと思われるようなインバリアントに関するとてもよいアイデアを持っている．この知識により，適当な初期重みを定義することができる．(例えば，インバリアントに興味のないプレースには 0 の重みを付ける等)

CP-nets および他の種類のハイレベル・ペトリネットに対しては，全ての p インバリアントを自動的に検出することは可能である．(これは色々な方法で実現可能である．例えば，ハイレベル・ネットをプレースとトランジションのネットに展開して，ガウス消去法を適用する等) しかしながら，そのような基底を見つけることは，しばしばあまり役に立たないことがある．その理由は，基底から所望する p インバリアントをいかに得るかを示す実質的，理論的なガイドラインが存在しないからである．こういう理由で Design/ CPN Palette は全ての p インバリアントの基底の計算は行っていない．

4.6 トランジションから実行コードへの変換

階層的 CP-net の中の各々のトランジションには CPN ML (最新のバージョンでは他の言語 C, Pascal 等でも可) で書かれたコードセグメントを付加することができる．これにより，システムのシーケンシャルな部分は Standard ML で記述し，制御構造は CP-net でプログラムするソフトウェアを生成することができる．そのプログラムはコードセグメント内の入出力コマンドにより，他のプロセスと通信することもできる．

プログラムが解析され，Design/CPN シミュレータおよび Design/CPN Palette の中のその他の解析ツールによりデバッグされた時，それらは実行可能な ML コードにマッピングすることができる．ML コードには，コードセグメントおよび図式的な情報を持たない CP-net の概略表現が含まれている．さらに ML コードは簡単なシミュレータを持ち，概略的な CP-net を実行し，発火トランジションに当るコードセグメントを起動する．

4.7 Design/CPN の使い心地？

本研究室では，使いやすさの一つの評価としてペトリネットの基礎知識のない学生にこのツールを用いたシステム開発をさせてみた．その結果，個々のツールとしての使い心地は明瞭で使いやすいが，ペトリネットをベースとしているため，ネットに付加する属性や変数定義の解釈が容易に行えなかったと言っている．なお，彼は今までこのようなシステムのモデル化の経験さえもなかったことも解釈を困難にした一因であった．

このことから，一般のソフトウェア開発に受け入れやすくする改善点はネットの解釈を助けることであると考えられる．

5 まとめ

以上，いくつかのネットワークツールの概要とそのうちのひとつである Design/CPN の基本仕様について述べた．これらのソフトウェアはすべてペトリネットをモデルとして採用していることから，並列分散システムのシミュレーション，プロトコル記述，に適したものであり，ネットワークの構造解析が可能なシステムである．つまり，これらの各種のツールは対象モデルの解析能力を十分にそなえているといえる．しかし，これらはペトリネットという特別な言語の知識を必要とし，ソフトウェアの生産現場で素直に受け入れられるとは言い難い．けれどもその応用分野はまだ未開発であり，図式的なもの以上に言

語的なインタフェースを整備すれば，ソフトウェアのみならず，様々な可能性を秘めていると推測される．今後さらに，これらの可能性を拡大する努力が必要であると思われる．

参考文献

[1] 翁長 健治, 情報システム構築の新しいパラダイムを求めて, ネット指向ソフトウェア設計技術に関するチュートリアル講演論文集, pp.1-11 (1992).

[2] G.D.Bergland [妹尾 稔 (訳)], 構造化設計法, ソフトウェア・ツール (bit 増刊), Vol.14, No.3, pp.221-246, 共立出版, 東京 (1982).

[3] 有澤 誠 ほか, マイケル ジャクソンのシステム開発法, bit, Vol.22, No.1, 共立出版, 東京 (1990).

[4] 日経コンピュータ, 導入始まるアッパー CASE ツール, 日経コンピュータ, 日経BP, 1992.3.9, No.3 (1992).

[5] R.Di Giovanni, "Petri Nets and Software Engineering: HOOD Nets," Proc.of 11th ICATPN, pp.123-138 (1990).

[6] G. Chiola, "GreatSPN 1.5: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets," Tool Description, PNPM'91 (1991).

[7] M. Ajmone Marsan et al., "TOPNET: A Tool Based on Petri Nets for the Simulation of Communication Networks," Tool Description, PNPM'91 (1991), also in IEEE Journal on SAC, Vol.8, No.9, pp.1735-1747 (1990).

[8] P.Parent, O.Tamir, "Voltaire: a Discrete Event Simulator," Tool Description, PNPM'91 (1991).

[9] F.Archetti, et al., "A Software Environment for the Generator, Representation and Analysis of PN based-Models of Manufacturing," Tool Description, PNPM'91 (1991).

[10] J.Billington, et al., "FORSEE," Tool Description, PNPM'91 (1991).

[11] J.Couvillion et al., "Performability Modeling with UltraSAN," Proc. of PNPM'91, pp.290-299 (1991).

[12] D.A.Marca and C.L.McGowan, "SADT," McGraw-Hill, New York (1988).

[13] E.Yourdon, "Managing the System Life Cycle," Yourdon Press (1982).

[14] D.Harel, "Statecharts: A Visual Formalism for Complex Systems," in Science of Computer Programming, Vol.8, North Holland, pp.231-274 (1987).

[15] K.Jensen, "Coloured Petri Nets. A Way to Describe and Analyse Real World Systems- Without Drowning in Unnecessary Details," Proc.of the IEEE 5th Int. Conf. on System Eng., pp.395-401 (1987).

[16] K.Jensen, "Coloured Petri Nets," in W.Brauer et al (eds.) Lecture Notes in Computer Science, Vol.118, Springer-Verlag, pp.327-338 (1981).

[17] K.Jensen, "Informal Introduction to Coloured Petri Nets," in ETACS Monographs on Theoretical Computer Science, Springer-Verlag (to be published).

[18] H.J.Grenrich and K.Lautenbach, "System Modelling with High-level Petri Nets," Theoretical Computer Science, 13, pp.109-136 (1981).

[19] H.J.Grenrich, "Predicate/Transition Nets," in W.Brauer et al (eds.) Lecture Notes in Computer Science, Vol.254, Springer-Verlag, pp.207-247 (1987).

[20] K.Jensen, "An introduction to high-level Petri Nets," Proc. of the IEEE 5th Int. Conf. on System Eng., pp.395-401 (1987).

[21] P.Huber et al, "Hierarchies in Coloured Petri Nets," 10th Int. Conf. on Application and Theory of Petri Nets, Bonn (1989).

[22] P.Huber, "Hierarchies in Coloured Petri Nets," Spec. for CPN Palette-Part2, Meta Software, Cambridge (1988).

[23] K.Jensen, "CPN ML," Spec. for CPN Palette-Part1, Meta Software, Cambridge (1988).

[24] A.Wikstrom, "Functional Programming Using Standard ML," Prentice-Hall (1987).

[25] R.Harper et al, "Standard ML," Tech. Report ECS-LFCS-86-14, University of Edinburgh, Edinburgh (1986).

[26] K.Jensen and S.Christensen, "The CPN Editor," Spec. for CPN Palette-Part3, Meta Software, Cambridge (1988).

[27] K.Jensen and S.Christensen, "The CPN Simulator," Spec. for CPN Palette-Part4, Meta Software, Cambridge (1988).

[28] J.Snow and R.Albright, "Design/IDEF User's Manual," Meta Software, Cambridge (1987).

[29] R.Nortin et al, "Design/OA User's Manual," Meta Software, Cambridge (1988).

[30] K.Albrecht et al, "Design/CPN: A Tool Package Supporting the Use of Colored Petri Nets," Meta Software, Cambridge (1989).