解　説

ネット指向パラダイムを求めて

5. 分散処理システムの開発とネット理論†

青　山　幹　雄††

1. はじめに

近年のプロセッサ技術と通信技術の発展にともない，情報処理システムのアーキテクチャは集中処理から分散処理へと急速に移行している．しかし，分散処理システムは，その静的構造のみならず動的挙動が複雑であり，従来の開発手法では，生産性，品質の両面で限界がある．最近，この領域は，**分散処理ソフトウェアの工学 (Distributed-Software Engineering)** と呼ばれ，多数の研究者の注目を集めている[53]．

ペトリネットなどのネット理論に基づくモデルは，システムの動的挙動の表現に優れていることと理論に基づく解析力の高さから，ソフトウェア，ハードウェア，外部環境を含む分散処理システムの統一したモデル化に適している．本稿では，ペトリネットによる分散処理システム開発の方法と適用を解説する．

まず，**2.** で分散処理システム開発の問題点をまとめ，この問題点に対しペトリネットなどのネット理論に基づくモデルの位置づけを **3.** で述べる．**4.** はペトリネットに基づく開発方法とプロセスをアプローチ，モデル化，設計，検証について述べる．**5.** は具体例に基づき実際的な開発方法を紹介する．**6.**，**7.** はペトリネットによる開発事例とペトリネットを用いた開発支援環境を概観する．**8.** は，ほかの開発方法との関係を論ずる．最後に，今後の課題とまとめを述べる．

2. 分散処理システムの開発における問題

図-1 に分散処理システムの開発プロセスを示す[53]．設計と試験工程で，集中処理システムの開発との違いがある．

分散処理システムの開発では，モデル化，設計，検証のいずれの工程においても，開発方法が十分確立されているとは言い難い状態にある．特に，実際の開発現場においては，従来の集中処理システムの方法に実務経験からの工夫を加えて開発が進められている状況にあり，分散処理システムの特性を明確にモデル化し，かつ，体系的な開発方法の開発と適用が望まれる．

以下では，分散処理システム開発のモデル化，設計，検証の三つのプロセスにおける問題点を抽出し，この問題点から開発方法に対する要求事項を明らかにする．さらに，ペトリネットがそのような要求条件にどうアプローチするかを示す．

2.1 モデル化の問題

対象システムの特性を形式的に表現できるモデルと，そのモデルに基づく設計方法が求められている．特に，次の三つの側面を明確に，かつ分かりやすく記述する必要がある．

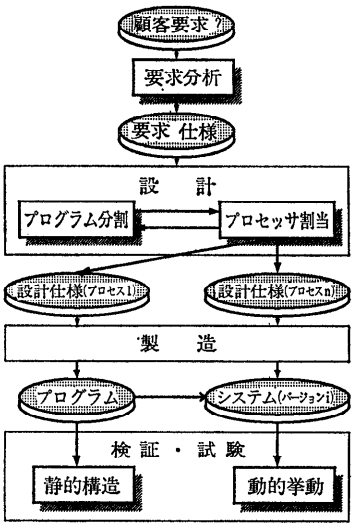


図-1　分散処理システムの開発プロセス

† Application of Net Theory to Developping Distributed Systems by Mikio AOYAMA (Business Switching Systems Division, Fujitsu Limited).

†† 富士通(株)複合交換機事業部

（1） **機能の側面：** システムはどんな機能要素から構成されているか？ また，その入出力はなにか？

（2） **構造の側面：** データなどのシステムの構成要素の構造と各要素間の関係がどうなっているか？

（3） **挙動の側面：** 各機能の実行はどう行われるのか？

分散処理システムでは，システムの**挙動**の側面が設計の重要な課題となるので，モデルが，次のような挙動の特性を表現できる必要がある．

- 並列性（Concurrency）

複数のプロセスが並列に処理を実行する．

- 非同期性（Asynchronism）

複数のプロセス間が非同期に処理を実行する．

- 非決定性（Non-Determininancy）

プロセスの実行がその動的状態などにより異なる．

このほか，通信システムや航空管制システムなど，実時間性を要求される分散処理システムは，**実時間分散処理システム（Real-Time Distributed Systems）**と呼ばれ，システムの時間的特性が重要な要求条件である[12]．また，航空管制システムや医療システムなどの**高信頼性システム（Safety-Critical Systems）**では，プロセッサや通信システムなどのシステム構成要素の障害に対するシステム全体の信頼性が重要な要求条件となる．このようなシステムの開発では，信頼性を考慮したシステム全体の統一的なモデルが望まれる．

2.2 設計の問題

モデルに基づき，システムの最適な構造を決定する．分散処理システムでは，図-1 に示すように，集中処理システムにおけるプログラム分割に加え，分割したプログラム（プロセス）をプロセッサに割り当てる**プロセッサ割当（Allocation）**とプロセッサ間の通信の問題を解決する必要がある．

2.3 検証の問題

開発プロセスの早期にシステムが要求仕様を満たすかどうかを検証することが望まれている．また，システムが大規模でかつ複雑になるにつれ，その試験コストは絶対額とともに総開発コストに占める比率も増大しているので，システムの効率的な検証，評価により総開発コストの削減が期待できる．

検証の対象は大別して次の三つである．

（1） 静的構造の検証

システムの構造がある特性を満たしているか？ 静的構造の特性としては，**完全性（Completeness）**，**一貫性（Consistency）**，**無曖昧性（Un-Ambiguity）**などが知られている．

（2） 動的挙動の検証

システムがデッドロックなどに陥ることなく，正しく動作するか？ など動的挙動の特性としては，デッドロックがないこと（Deadlock Freeness），バッファがオーバフローしないことなどがある．

（3） 性能の検証

システムがある性能要求条件を満たすこと．特に，実時間システムやオンライン・トランザクション処理システム（OLTP）などでは，システムの重要な要件である．性能評価の指標としては，スループットなどの定常的な処理能力と応答時間のような過渡的な処理能力の二種類がある．

分散処理システムでは，各プロセッサ内の**情報処理**とともにプロセッサ間の**通信処理**も検証，試験する必要があるため，検証と試験のコストが増大する傾向にある[55]．特に，通信処理では，実時間性を含む正常性を検証する必要がある．しかし，微妙なタイミングによりシステムの挙動が非決定的であるため，**再現性（Reproducibility）**に乏しい．このように，分散処理システムの試験では，並列処理性，非同期性，非決定性の問題が複雑に絡み，検証や試験がきわめて困難である．実際の分散処理システムのデバッグでは，問題が再現できればデバッグの半分は達成されたと言われるゆえんである．

3. 分散処理システム開発とペトリネット

分散処理システムのモデルは，表現形態と意味の観点から**図-2** に示すように分類される．

3.1 表現形態からみたペトリネット

（1） 図形モデル

ネット理論などに基づく図形やアイコンなどにより構成される視覚的モデル．

（2） テキストモデル

テキスト形式の仕様記述言語やプログラム言語を拡張した言語などのモデル．

（3） ハイブリッドモデル

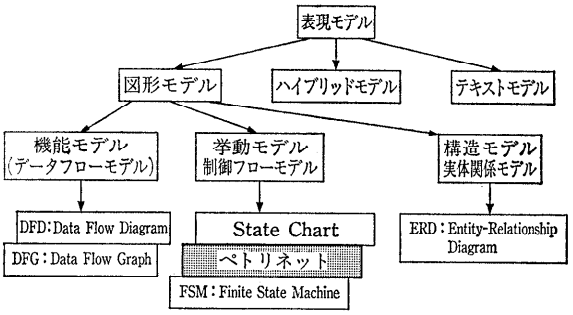


図-2 システムのモデル化方法

図形とテキストを統合したモデル.

図形モデルとテキストモデルにはそれぞれ長所，短所があり，目的に応じて相互に補完して利用すべきであろう．本稿では，ペトリネットなどのネット理論に基づく図形モデルに焦点を当てて解説するが，分散処理システムのモデルとして，テキストモデルも活発に研究されている．たとえば，Hoare の CSP (Communicating Sequential Processes) や Milner の CCS (Calculus of Communicating Systems) が知られている．最近，時制論理 (Temporal Logic) に基づくモデルも活発に研究されている[30),35)].

3.2 図形の意味からみたペトリネット

図形モデルは，モデルが表現するシステムの機能，構造，挙動の特性から分類できる．

システムを実際に実現し，実行する前にシステムの挙動を知ることは難しい．しかし，システムの動作は，外部入力に対する応答などシステムの要求仕様に直接関わる重要な特性である．このため，**システムの挙動モデル**を早期に構築することが必要である[24)]．ペトリネットは，並列性，非同期性，非決定性などとともにシステムの性能もモデル化できるので，分散処理システムの挙動モデルとして適している．

3.3 ネットの意味からみたペトリネット

ネットに基づくモデルは，ネットの意味づけの観点から，次の三つに分類できる．

（1） 制御フローモデル

制御のダイナミックスを中心に考えるアプローチで，制御フローをネットにより表現する．

（2） データフローモデル

データを中心にシステムをモデル化するアプローチであり，データフローをネットにより表現する．データ間の同期条件により制御のダイナミックスを表す．

（3） ハイブリッドモデル

制御フローとデータフローを統合して表示することにより，モデル化能力を高めるアプローチ．

3.3.1 制御フローモデル

制御フローモデルとして，次のようなモデルがある．

（1） **状態遷移モデル (FSM: Finite State Machine)**[8)]．FSM に基づく多数の表記法が提案されている．たとえば，通信システムの記述では，CCITT により標準化された SDL (Specification and Description Language) が普及している．また，複数の FSM から成るモデルとして，CFSM (Communicating Finite State Machines)[9)] や Venn 図表を導入して階層化する **State Chart**[23)] などが提案されている．

（2） **ペトリネット (Petri Net)** とその拡張モデル．ペトリネットは FSM でモデル化する状態遷移とその遷移を引き起こすアクションの両方を一つのモデルで表現できる点で FSM を包含している．さらに，並列処理のモデル化や階層的なモデル化も可能である．

3.3.2 データフローモデル

データフローダイアグラム (DFD: Data Flow Diagram) が，米国を中心として，ビジネスアプリケーションの要求分析で広く用いられている[17)]．一方，DFD とは異なるモデルであるが，データフローアーキテクチャを採るシステムのモデルとして**データフローグラフ (DFG: Data Flow Graph)** も提案されている[20)]．

3.3.3 ハイブリッドモデル

このモデルはまだ少ない．実時間システムの要求分析のために DFD に制御フローを記述可能とした，いわゆる，**リアルタイム DFD** やペトリネットとデータフローを統合した **MPN (Modified Petri Net)**[65)] がある．MPN の拡張として筆者らは DISCOL (DIStributed Communication-Oriented Language)[4)] を開発している．

3.4 ペトリネットによる開発方法の特徴

ペトリネットによる分散処理システムの開発方法は次のような特徴がある．

（1） 動的挙動の形式的なモデル化

分散処理システムの挙動モデルとして必須で

ある，複数プロセッサ間の並列性，非同期性，非決定性を厳密にモデル化できる．さらに，モデルの構成要素が簡潔である．

（2） 視覚的表現

ネット構造によりシステムの構造を視覚的に表現できるため，モデル化や解析において直観的理解を促す．したがって，設計者相互のみならず異なるグループ（設計者や分析者）間の共通モデルとして利用できる．

（3） ネット理論に基づくシステム解析

ネット理論に基づき，デッドロック，バッファオーバフローなどの構造特性や，スループットなどの性能特性など，システムのさまざまな動的特性の検証や定量的評価ができる．

（4） 実行可能な仕様（Executable Specification）

ペトリネットにより設計したシステムの一部，あるいは全体をワークステーションなどの上で直接実行し，構造や挙動を確かめながら設計を進める**プロトタイピング**を支援する．

（5） 定量的評価に基づく体系的な開発方法の支援

ペトリネットの解析力や実行可能性により，システムの挙動や性能の定量的評価データに基づき開発を体系的に進めることができる．

一方，問題点もある．たとえば，初期のペトリネットでは，大規模システムをモデル化するための抽象化の枠組みが弱かった点や解析の計算時間がきわめて大きい点である．しかし，近年，このような問題に対する精力的な研究が進められ，著しい改善がみられる．

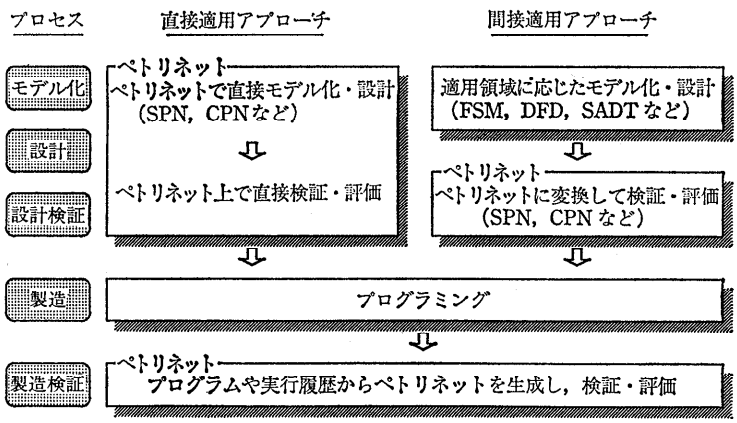


図-3 ペトリネットの適用アプローチ

4. ペトリネットによる開発方法

ペトリネットによる開発方法を，アプローチ，モデル化，設計，検証に分けて概観する．

4.1 アプローチ

ペトリネットをシステム開発に適用するアプローチとして，**図-3** に示す二つがある．

（1） 直接適用アプローチ

システムを直接ペトリネットによりモデル化し，設計，検証をペトリネット上で行う．

（2） 間接適用アプローチ

適用領域ですでに確立しているモデル化方法や既存のモデルをそのまま利用する．ペトリネットの解析力や実行可能性を活用するため，既存のモデルからペトリネットに変換するなどしてシステムの検証，評価のみをペトリネットにより行う．

これらのアプローチには，それぞれ，長所と短所があり，その比較を**表-1** に示す．

4.2 モデル化

対象システムをペトリネットによりモデル化する場合，次の点に留意する必要がある．

表-1 ペトリネット適用アプローチの比較

比較項目	直接適用アプローチ	間接適用アプローチ
表現力	○並列性などの表現力が活用可能 ×データ構造などの表現が不可能	○適用領域に応じたモデル化方法を利用 ×モデルにより並列性などの表現が不可能
解析力	○ペトリネットの解析力が十分活用可能	？既存モデルの表現力や変換可能性による
適用性	×ペトリネットのモデル化の習得が必要	○既存の開発方法の拡張
適用領域	システムが比較的小規模の場合や既存方法ではモデル化が困難な場合	すでにモデルが確立していたり存在している場合にシステムの挙動などの検証・評価を行う

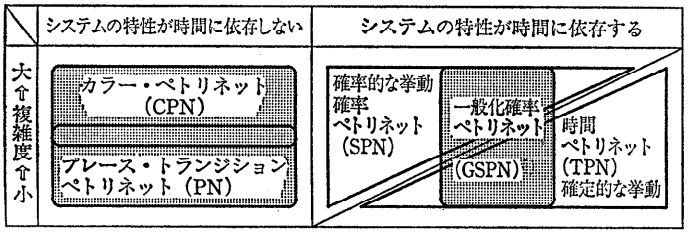


図-4 ペトリネットモデルの選択指針

（1） 対象システムは何か？

対象システムの特性やモデル化すべき側面を明確にする．ペトリネットは制御フローに基づくモデルであるので，ビジネスアプリケーションなどの**変換処理システム (Transformation System)** のモデルには適さない．

（2） 目的は何か？

大規模システムの開発では，システムの複雑で多様な特性を単一のモデルで表現することは困難である．特に，分散処理システムでは，モデルの表現がきわめて複雑になる傾向があるため，モデル化の狙いを絞り，記述を簡単にすることが望ましい．プロセッサの機能，負荷分担を決定するアーキテクチャ設計を行うのか，デッドロックなどの挙動の解析を行うのか，システムの性能評価を行うのか，などの狙いにより適用するモデルを選択する．このため，**図-4** に示すように，時間的特性と対象システムの複雑度に応じてモデルとして用いるペトリネットを使い分ける．主なペトリネットのクラスとして，次の三つがある．

①PN（プレース・トランジション・ネット）: プレースとトランジションからなり，最も広く用いられているペトリネット[32),44),49),50),54)]．

②確率ペトリネット (SPN: Stochastic Petri Net): トランジションの発火が指数分布関数に従い，ランダムな事象をモデル化できるため，性能評価などのモデルに適用する[36),37)]．

③高水準ペトリネット (HPN: High-Level Petri Net): カラーペトリネット (CPN: Colored Petri Net)[28)] や**述語／トランジション・ネット (PrT: Predicate/Transition Net)[21)]** は，同一構造のネットを一つのネットにたたみ込みトークンの**色**で識別できるため，複雑なシステムを抽象的にモデル化できる．PN をアセンブラ言語とすれば，HPN は，抽象データ型を支援する高水準言語に相当する．

（3） どうモデル化するか？

ペトリネットによりシステムをモデル化するには，プレースやトランジションの意味を対象システムと対応させて定義しなければならない．ペトリネットのシステム的な解釈の主な例を**表-2** に示す．

さらに，**図-5** は，基本的なシステム構成要素を PN でモデル化した例を示す．Fork と Join は，それぞれ，プロセスの生成と終了を行う．**競合 (Conflict)** は，プロセス p_2 が二つのトランジション t_1 と t_2 に共有されているため，いずれか一方が発火すると他方が発火できない構造である．**混乱 (Confusion)** は，実行順序による非決定性をモデル化している．独立したトランジション t_1 と t_2 の発火の順序がトランジション t_3 の発火に影響を及ぼす．t_1 が t_2 より先に発火すると，トランジション t_3 は発火可能となるが，t_2 が先に発火すると，t_3 は発火できない．

図-6 は CPN によりプログラムの基本的な制御構造を表現した例である[28)]．このほか，Ada プログラムのランデブ (Rendezvous) での種々の同期機構を PN でモデル化した例が文献 34) にある．

実際にシステムをペトリネットでモデル化するには，対象システムの理解とペトリネットによるモデル化方法の理解が必要であり，例題や事例などにより習熟する必要がある．さらに，モデルの良否は，以後の設計や検証の生産性と品質に大き

表-2 ペトリネットのシステム的解釈

入力プレース	トランジション	出力プレース
入力データ	処理	出力データ
バッファ	プロセッサ	バッファ
捕捉リソース	タスク・ジョブ	解放リソース
プリコンデション	イベント	ポストコンデション
論理式（前提）	節 (Clause)	論理式（結論）

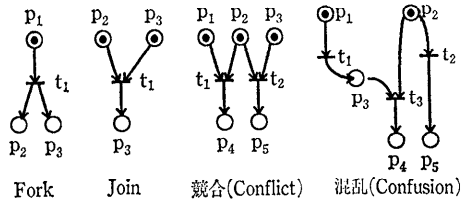
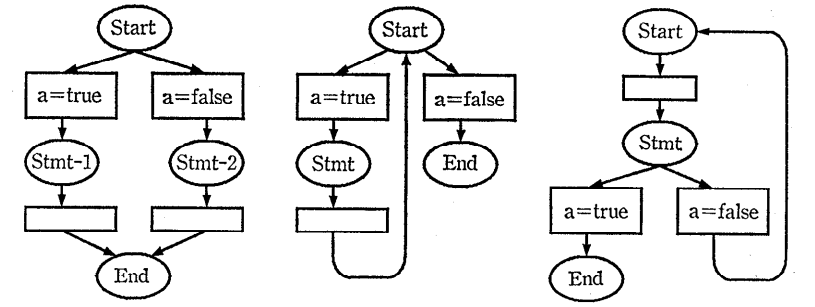
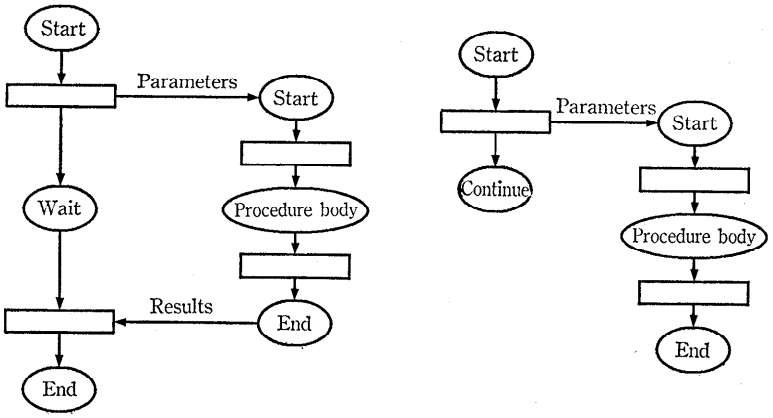


図-5 ペトリネットによる分散処理の基本要素モデル



IF a THEN Stmt-1 ELSE Stmt-2　　WHILE a DO Stmt　　REPEAT Stmt UNTIL a



Procedure Call (復帰型)　　Procedure Invocation (非復帰型)

図-6 プログラム制御構造の CPN 表現

く影響する．たとえば，Marsan らは，システム構造の意味を考慮することにより，モデルの記述を大幅に簡略化できる方法を示している[38]．最も簡略化したモデルでは，簡略化しないモデルに比べ状態空間の大きさが 1/4,000 となった．

4.3 設　計

一般に，**図-7** に示すトップダウンによる段階的詳細化とボトムアップによるシステム構築とが組み合わされた次の三段階の設計プロセスに従うことになろう．もちろん，必要であれば，このプロセスは，最適な設計が得られるまで繰り返し実行される．

①システムを構成するオブジェクトとその間の関係の抽出

②各オブジェクトの内部構造の設計

③各オブジェクトを統合したシステム全体の設計

FMS (Flexible Manufacturing System) の事例[11]により，設計を行う方法を簡単に紹介する．具体的な設計事例は **5.** を参照願いたい．

（1） システムを構成するオブジェクトとその間の関係の抽出

FMS の主要なオブジェクトは，部品を保管する倉庫，部品を運ぶ搬送車，加工を行う加工機械，部品の配送を制御するディスパッチャである．さらに，各オブジェクト間の関係をネット構造とその上を流れるトークンで記述する．**図-8** に，搬送車と加工機械の外部設計を示す．

（2） 各オブジェクトの内部構造の設計

各オブジェクトの内部構造，すなわち状態遷移を設計する．搬送車と加工機械の例を，**図-9** に示す．

（3） 各オブジェクトを統合したシステム全体の設計

各オブジェクトを結合し，全体システムを構築する．**図-10** に全体システムの構成を示す．

この例では，搬送車や加工機械などのオブジェクトのレベルと各オブジェクトの内部構造のレベルとの二階層にシステムを分割している．

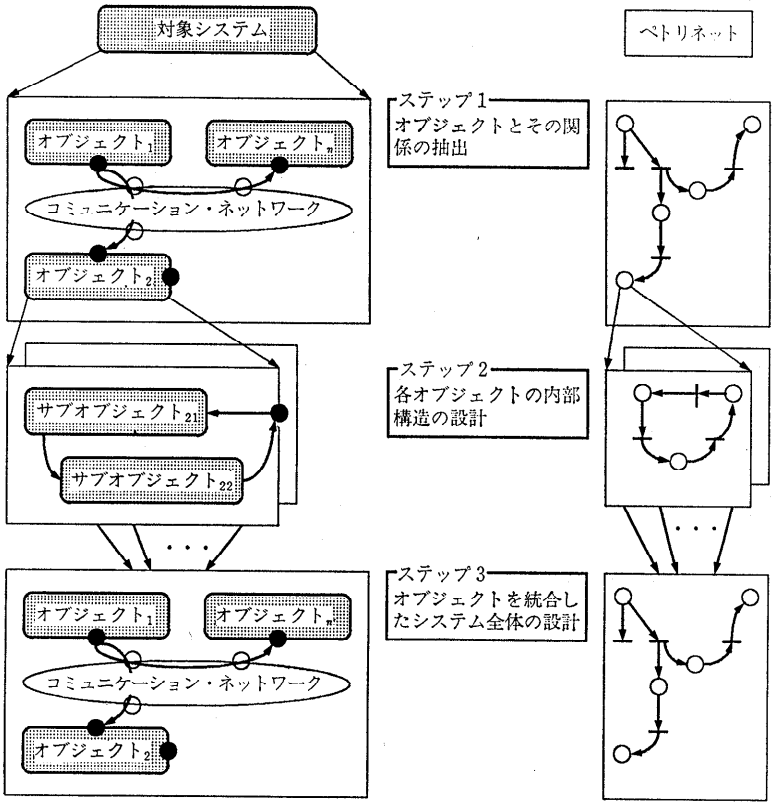


図-7 ペトリネットによる設計プロセス

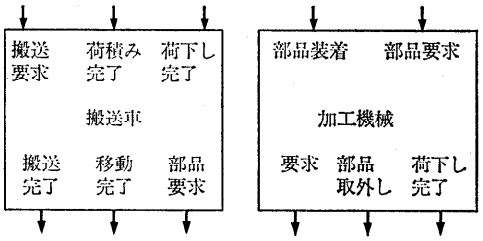


図-8 オブジェクトの外部設計

4.4 検　証

4.4.1 検証方法

ペトリネットを分散処理システムの検証，試験に適用する方法を図-11 に示す．

要求分析や設計工程では，システムを実現する前に，ペトリネットによる記述に基づき挙動や性能などを検証する．一方，システムを実現した後に，ソースプログラムや実行履歴からペトリネットを生成し，システムの挙動を分析する方法が提案されている．これは，**リバースエンジニアリング (Reverse Engineering)**[14] の一種と考えられる．**設計情報のリカバリ (Design Recovery)** と対比すれば，**挙動情報のリカバリ (Behavior Recovery)** と呼べよう．**この目的は，挙動のリエンジニアリング (Behavior Re-Engineering)** にある．

4.4.2 ペトリネットによる特性の解釈

ペトリネットで解析できるネット構造の基本的な特性がシステムの特性としてどう解釈できるかを理解しておくことが重要である．ネットの特性は，初期状態，すなわち，初期マーキングに依存する特性と依存しない特性に分類される．前者は**マーキング依存の特性 (Marking-dependent property)** と呼ばれ，後者は**構造的特性 (Structural property)** と呼ばれる[44]．

(1) **可到達性 (Reachability)**

与えられた二つのマーキング M_0, M_1 間の遷移($M_0 \rightarrow M_1$)が可能か否かを決定する．後述するように，可到達性は，システムの構造を解析する最も基本的な特性である．可到達性は**決定可能 (Decidable)** であり，**可達木 (Reachability tree)** を生成し，その上で**状態探索 (State exploration)** により解析される．これを実行するソフトウェアツールも提供されている．しかし，一般に，プレースの数に対し

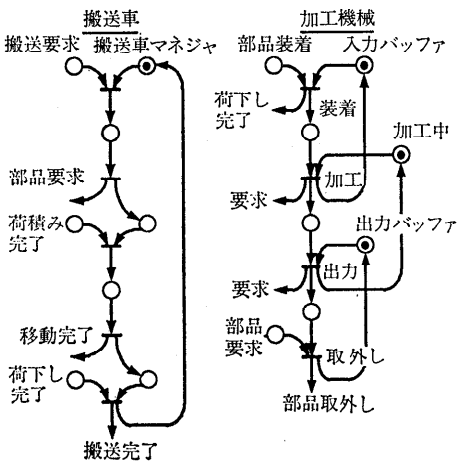


図-9 オブジェクトの内部設計

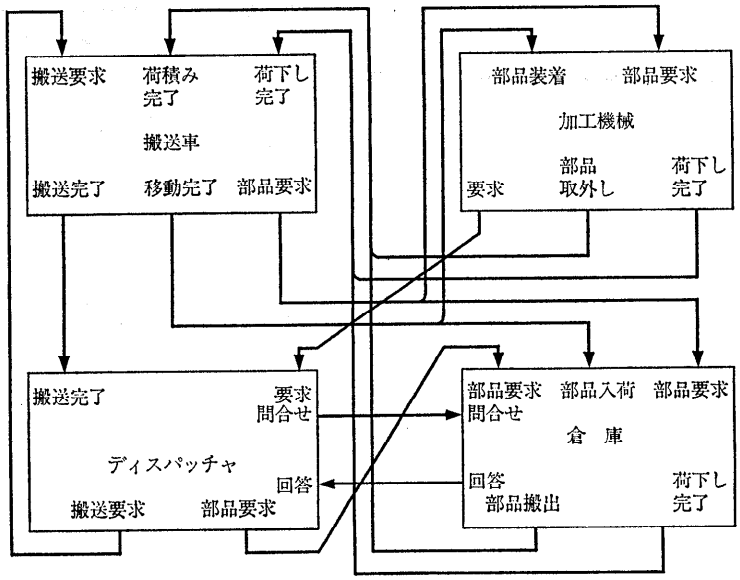


図-10　合成した FMS の全体システム

指数関数的計算時間とメモリが必要であることが証明されている．この計算量の問題が大規模システムにおける可到達性解析を困難にしている．計算量削減のために，階層的記述やペトリネットを簡略化する方法など，種々の方法が提案されている．

（2）**有界性（Boundedness）**

各プレースのトークン数が，常に越えないある上限値が存在するか否か．システムとしては，たとえば，必要なバッファの長さやタスク数に上限があるか否かを意味する．

（3）**安全（セーフ）性（Safeness）**

有界なペトリネットで，その上限値が1の場合．たとえば，シングルタスクで実行される場合．

（4）**活（ライブ）性（Liveness）**

各トランジションが発火可能であり，次のマーキングへ遷移可能であるか否か．発火不可能なトランジションはシステムとしてはデッドロックを意味するので，活性とはデッドロックがないことである．実際のシステムでは，もっと緩い活性であってもよい場合があるので，より緩和された活性も定義されている．

（5）**持続性（Persistence）**

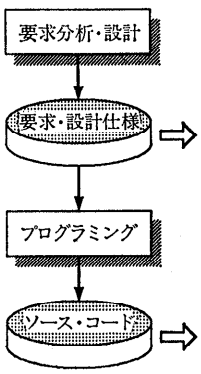
持続性とは，任意の二つのトランジションの間で，一方の発火が他方に影響を与えないことである．これは，競合（Conflict）の有無を意味する．

（6）**同期距離（Synchronic distance）**

ペトリネットにおける二つのトランジションの相互依存性として定義され，システムにおける二つのタスク間の相互依存性を示す尺度である．

（7）**公平性（Fairness）**

複数タスク間の相互依存性のもう一つの尺度として，トランジションの遷移の公平さを示す尺度が種々提案されている．たとえば，**有界公平性（Bounded fairness）**とは，トランジション t_1 と t_2 の間で，一方を発火せずに有界の発火回数で他方を発火可能な場合に，t_1 と t_2 は有限公平であるという．公平性のほうが，同期距離より実際的であると言われる．すなわち，同期距離∞にもかかわらず，公平性が有限の場合がある．



仕様の検証	
構造などの特性解析	可達木解析などのネット理論に基づく仕様段階での解析
シミュレーション	仕様をペトリネットシミュレータなどで疑似実行し動的挙動の視覚化や性能評価

プログラムの検証	
構造などの特性解析	プログラムからペトリネットに変換し，仕様の解析と同様にネット理論に基づき解析
実行履歴の解析	プログラムの実行履歴からペトリネットを生成しペトリネット上で解析・評価

図-11　ペトリネットによる検証方法

5. ペトリネットによる開発の実際

ペトリネットによる分散処理システムの具体的な開発方法を概観する．詳細は文献 6) を参照願いたい．

5.1 ペトリネットによる設計の実際

踏切の制御システムは，きわめて高い信頼性を要求される高信頼性 (Safety-Critical) システムである．この種のシステムの設計では，ハードウェアとソフトウェア，場合によっては人間のオペレーションをも含むシステム全体の統一的なモデルが必要である．Leveson らは，ペトリネットによりシステムの安全性を保証する設計，解析方法を提案している[33]．**図-12** では，踏切制御システムだけでなく，列車と遮断機を含む分散処理システム全体が単一のモデルで表現されている．

さらに，障害 (Failure) トランジションと障害 (Fault) プレースを導入して，システムのある部分の障害 (Failure) や誤動作もペトリネットでモデル化できる．図中，障害トランジション f_5 は踏切制御コンピュータからの誤った信号をモデル化している．したがって，プレース p_{14} のトークン数は障害の発生数となる．列車が接近中になると p_1 のトークンは p_9 へ移動し，遮断機が上がっていると，t_7 が発火可能となり，遮断機が下りる．ここで，障害トランジション f_5 が発火すると，遮断機を上げるトランジション t_6 が発火可能となり，誤動作を起こす．

図-13 は，図-12 に対し安全設計を施し，Fail-Safe となった踏切制御システムである．トランジション R_1 と R_2 は，それぞれ障害の検出と復旧を行う．プレース p_6, p_7 は，遮断機の状態をコンピュータの内部状態としてモデル化している．し

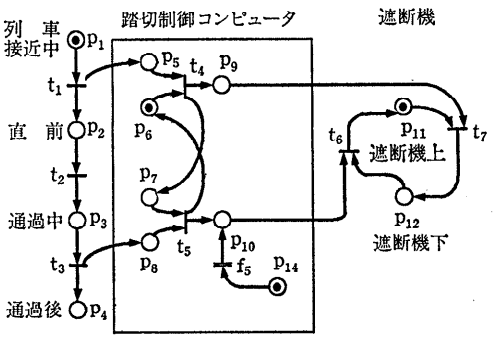


図-12 踏切制御システムのペトリネットモデル

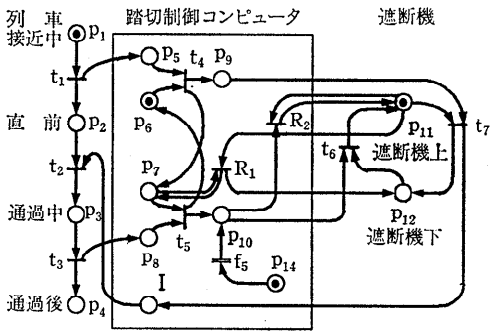


図-13 安全な踏切制御システム

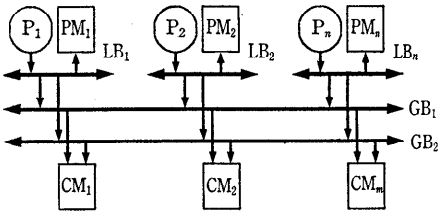
たがって，p_7 と p_{11} が一致することはシステムの状態として矛盾するので，障害として検出される．障害により遮断機が上がっている場合，最も安全な復旧機構として R_2 が遮断機を下ろす．

実際，図-13 の可達木を生成すると，このシステムが安全であることが分かる．このように，システムが安全であるための必要十分条件がその可達木で厳密に解析できる．

5.2 確率ペトリネットによる設計の実際

マルチプロセッサ分散処理システムの設計ではプロセッサとその間の通信システムのバランスがシステムの性能を決定する重要な要因である．このため，設計段階でシステムの性能を評価することが必要である．このようなシステムの設計では，プロセスやプロセッサ間の同期をモデル化する必要がある．同期のモデル化は，従来性能評価に広く適用されていた待ち行列モデルでは困難であるが，確率ペトリネット (SPN) やその拡張である**一般化確率ペトリネット (GSPN : Generalized Stochastic Petri Net)** では容易にできる[37]．さらに，SPN と**連続時間マルコフ連鎖 (Continuous Time Markov Chain)** とが対応することから，システムのスループットなどの定常状態の性能が行列方程式により代数的に求まるという長所がある．このような評価方法の開発は，分散処理システムの設計が定量的な評価データに基づき体系的にできる点で有意義である．

一例として，**図-14** に示すバス結合のマルチプロセッサシステムで5プロセッサ，三共通メモリ (CM)，二共通バス構成の GSPN モデルを**図-15** に示す．p_1 のトークンがプロセッサ数を，p_2 のトークン数が共通バスの本数を表す．**図-16** は，



P：プロセッサ，PM：個別メモリ，CM：共通メモリ，
LB：共通バス，GB：共通バス

図-14　バス結合のマルチプロセッサシステムの例

共通バスの数を固定（＝2）とし，$\rho=\lambda/\mu$（共通メモリのアクセス時間とプロセッサの処理時間の比）をパラメータに取り，プロセッサ数に対するシステム全体のスループットの評価結果を示す．

ここで，GSPN モデルの構造がプロセッサ数や共通バス数などの分散処理システムの複雑度によらないことに留意したい．

文献 22)では，非有界 (Unbounded) GSPN を用いて，マルチプロセッサによるロボット制御システムの設計例を示している．GSPN 上でのシミュレーションにより最適なプロセッサ数やメモリ容量が求まる．このほか，SPN や GSPN は各種の分散処理システムの性能評価に適用されて，大きな成果を収めている[16),37)]．

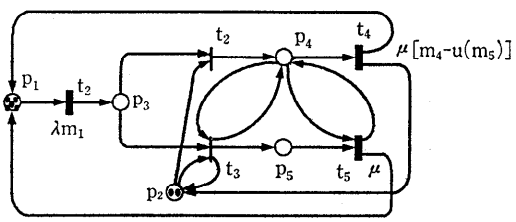


図-15　マルチプロセッサシステムの GSPN 表現

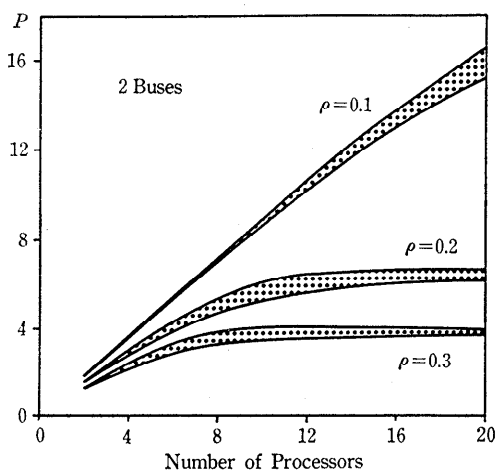


図-16　マルチプロセッサシステムの性能評価結果[37)]

5.3　ペトリネットによる検証の実際

分散処理システムの検証にペトリネットを応用する二つの方法を，Ada プログラムの構造解析[52)]と動的実行履歴の解析[26)]を例として示す．

5.3.1　Ada プログラム構造の静的解析

Shatz らは Ada プログラムからペトリネットを生成し，ペトリネットの可達木解析により，プログラムのデッドロックの有無などの特性を分析する方法を提案している．

プログラムの例を**図-17** に示す．三つのタスク T_1, T_2, T_3 が，それぞれ，エントリ E を介してランデブする．生成されたペトリネットとその可達木を，それぞれ，**図-18**，**図-19** に示す．

図-19 の可達木を見よう．このプログラムには二つのデッドロック状態（34 と 35）を含むことが分かる．状態 34 は，図-18 のペトリネットのプレース 6, 13, 11, 17 にトークンがあるマーキングに対応する．これは，タスク T_1 と T_2 は処理を完了し，タスク T_3 がタスク T_2 からの Acknowledge を待っている状態を意味する．

また，同様の方法により Occam プログラムを分析する方法も提案されている[64)]．

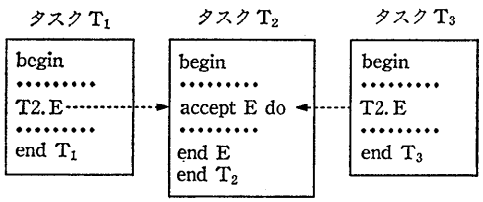


図-17　三つのタスクによる Ada プログラムの例

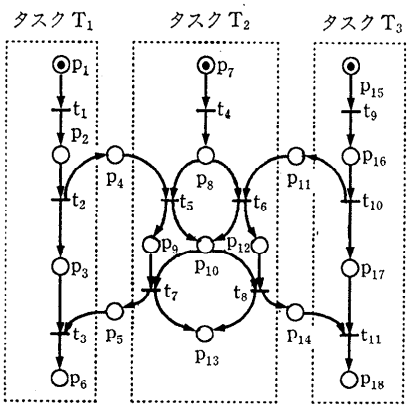


図-18　Ada プログラムのペトリネット表現

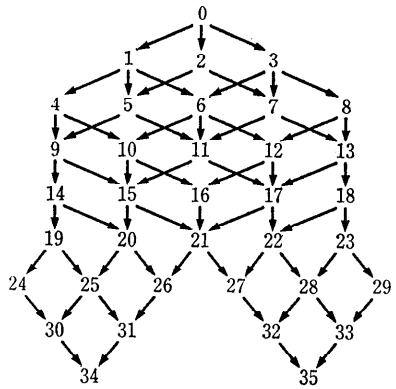


図-19 Ada プログラムの可到達木

5.3.2 プログラム実行履歴の解析

分散処理システムのデバッグでは，挙動の非決定性により問題の再現や解析が困難である．このため，分散処理システムのデバッグを支援する**分散処理システムデバッガ (Distributed Debugger)**[39] が提案されている．

分散処理システムデバッガの一種として，システムの実行履歴からペトリネットを生成し，その動的挙動をペトリネットで解析する方法が提案されている[4),26)]．

図-20 に解析システムの構成例[26]を示す．対象システムから収集したイベントの履歴をペトリネットに変換し，システムの外部仕様や設計仕様から生成した正常履歴と比較することによりシステムの動的挙動を検証する．

6. ペトリネットによる開発事例

ペトリネットによる分散処理システムの開発事例を**表-3** にまとめた．詳細は各参考文献を参照願いたい．もちろん，このほかにも多数の事例があり，文献 5) に開発事例のサーベイがある．また，ドイツの GMD による 1990 年末時点でのペトリネットに関する 4,000 件余りの論文リスト[48]があるので，参照されたい．

7. ペトリネットによる開発支援環境

ペトリネットを実際の分散処理システムの開発に適用するには，コンピュータによる開発支援環境が必須である．このため，ワークステーションやパーソナルコンピュータを利用した開発支援環境が開発されている．特に，最近のグラフィカル・ユーザ・インタフェース (GUI) の発展により，支援環境の機能は格段に向上している．このようなツールの中には，**CASE (Computer-Aided Software Engineering)環境**として大規模システム開発に利用しうるものも現れており，商用ツールも提供されている[6),31),54)]．DFD や木構造チャートなどを用いる CASE 環境と比較し，ペトリネットによる環境の特長は検証と実行にある．可達木による理論的な解析とシミュレーションによる評価を可能とする種々のツールが提供されている．

開発支援環境は，次のような機能を提供する．

(1) 設計：GUI を利用してペトリネットの作成，編集を支援する図形エディタ．

(2) 解析：可達木解析などによるペトリネットの構造解析．

(3) 実行：ペトリネットをワークステーション上でシミュレーションし，システムの挙動や性能を評価し，視覚的に表現する．

筆者らも，GSPN をベースとするペトリネットツール Integral を開発し，通信ソフトウェア開発への適用を試みている[4)]．**図-21** は Chiola らが開発した GSPN の開発支援環境 **GreatSPN**[15] を用いて交換サービスを記述した例である．

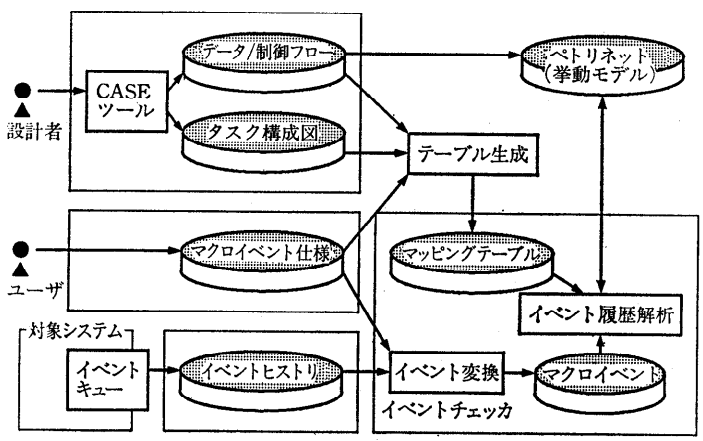


図-20 ペトリネットによる実行履歴解析システム

8. ほかの開発方法との関係

ペトリネットとほかのいくつかのモデルや開発方法との関係について概説する．

表-3　ペトリネットの適用状況

適用分野	適用アプローチ	モデル	参考文献
マルチプロセッサシステム	処理能力評価に基づくアーキテクチャ設計	GSPN	36),37)
	並列処理制御方式の設計と実装	PN	42)
	ディジタル信号処理プロセッサの制御方式	PN	10)
	Unbounded-GSPN によるロボットの制御方式	GSPN	22)
	ジョブフロー制御方式	SPN	56)
分散 DBMS	分散 DBMS の制御方法の解析・評価	PrT	45),61)
	分散 DBMS の問合せの高速化	PN	40)
	複数コピー間の一貫性の解析	SPN	18)
	分散データの障害復旧のモデル化と解析	CPN	1)
信頼性解析	システムの取替えを考慮した信頼性解析	PN	57)
	信頼性，安全性，可用性の解析，評価	SPN	2),3)
プログラム	Ada プログラムのモデル化と解析	PN	34),52)
並列論理プログラム	並列論理プログラムの簡略化と検証	PrT	43)
	高速推論の実現方法	PrT	46)
	充足性，証明可能性の検証	PrT	62),63)
	GHC の並列性の解析	PrT	25)
	ファジィ PN (FPN) による曖昧推論	FPN	13)

8.1 オブジェクト指向分析方法との関係

図-22 は，**2.** で述べたシステムの三つの側面をオブジェクト指向分析方法がどのように採り入れているかを示す．いずれも，システムの挙動の側面をモデル化するために FSM に基づいたモデルを採用している．特に，OSA では，挙動モデルとしてペトリネットの一種である State Net を採用している．また，Rumbaugh は，挙動モデルに関してプレース・トランジション・ネットに言及し，記述レベルが低い点を指摘している[51]．しかし，最近の高水準ペトリネットについての比較はない．今後，どのようなモデルがオブジェクト指向分析の挙動モデルとして使われるのか興味ある課題である．

図-23 は，OSA の State Net を用いて電話によるオーダエントリシステムをモデル化した例を示

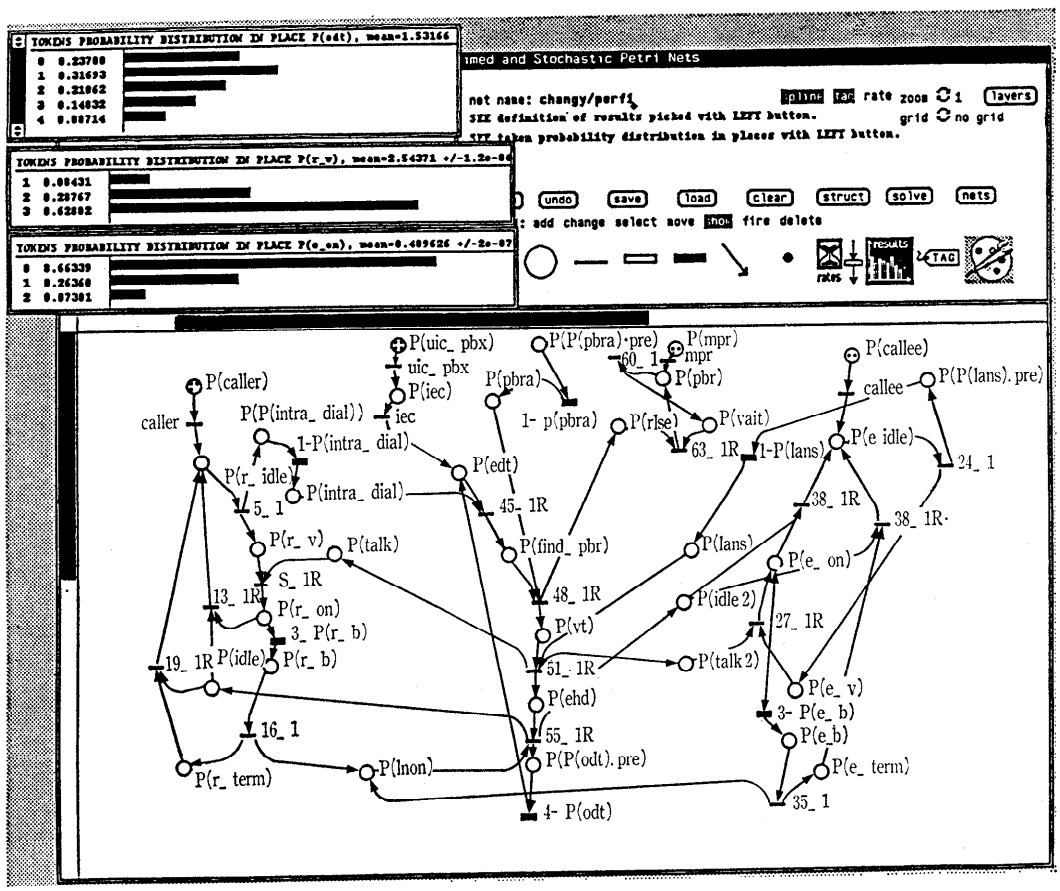


図-21　GreatSPN の画面例

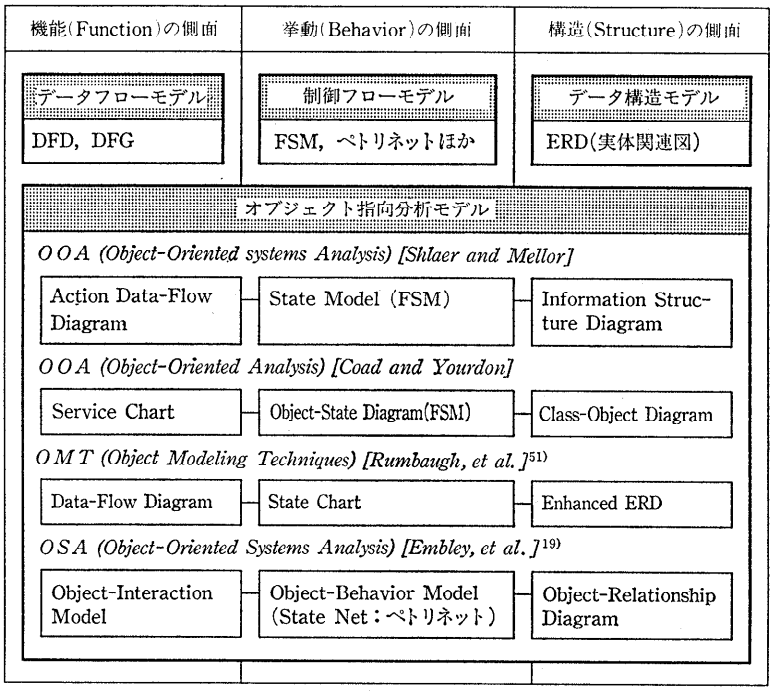


図-22 オブジェクト指向分析モデルとの関係

す[19]．楕円がプレースを，ボックスがトランジションを示す．ボックスの上段は遷移のトリガを，下段は遷移の動作 (Action) を記述する．左端の縦線はペトリネットのトランジションと同一の記法であるが初期遷移 (Initial Transition) と呼ばれ，状態遷移の起点を示す．

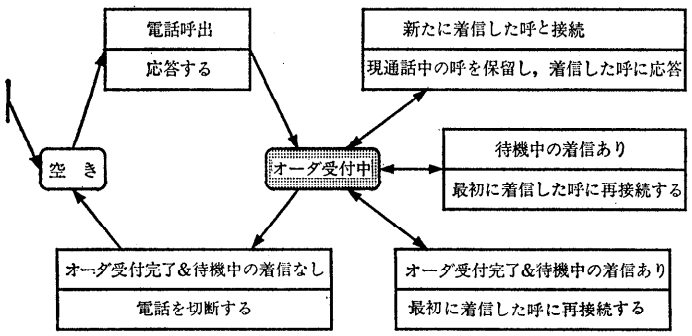


図-23 State Net の記述例

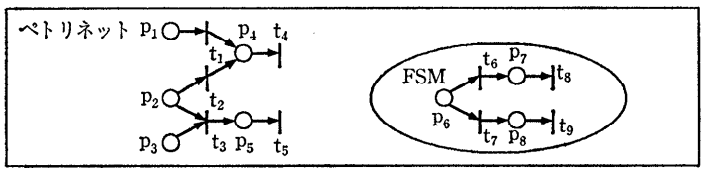


図-24 ペトリネットと FSM

8.2 ほかのネットモデルとの関係

8.2.1 ペトリネットと FSM

ペトリネットは FSM の拡張であることが示されている．すなわち，図-24 に示すように，FSM はペトリネットで各トランジションの入力プレース数＝出力プレース数＝1 とした特殊な場合である[44]．

図-25 は，二つのプロセス間の簡単なプロトコルを FSM とペトリネットにより記述した例である．ペトリネットの場合，状態遷移とそのアクションとの両方を記述するため，同期関係を FSM より明確に記述できる．しかし，記述の複雑度が増すという欠点もある．

8.2.2 ペトリネットとデータフローモデル

あるクラスのペトリネットと DFG が準同型 (Isomorphic) であることが示されている[29]．従来，DFG の形式的なモデル化や解析方法はあまり研究されていない．このため，ペトリネットに変換し

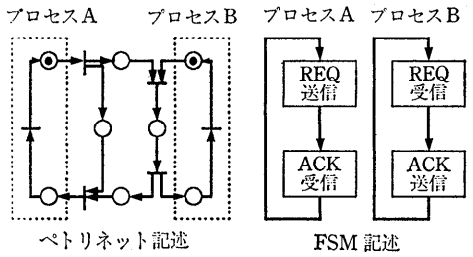


図-25 簡単なプロトコルの記述例

てシステムの特性を分析する方法が提案されている．Kavi らは，データトークンが意味付けをもたない DFG とペトリネットとの相互変換アルゴリズムを示している．

また，DFD の解析方法として，ペトリネットを利用する二つの方法が提案されている．一つは，DFD をペトリネットに変換し，ペトリネット上で解析する方法である．もう一つは，DFD の動的挙動モデルとしてペトリネットを想定し，DFD の各要素の動的実行規則を対応するペトリネットの動作として解釈する方法である．いずれも，ペトリネットのもつ動的挙動の表現力や解析力を活用している．

8.3 ペトリネットと時制論理

ペトリネットと時制論理とは，分散処理システムのモデル化や解析において相互補完的な位置にあることが指摘されている[27]．すなわち，ペトリネットはシステムの動的構造のモデル化に適しており，時制論理はシステムの特性の解析や制約条件の記述に適している．また，ペトリネットはシステムの内部状態の遷移として表現するモデルであるが，時制論理はイベントの系列として表現するモデルである．この二つのモデルを組み合わせて分散処理システムの開発に適用する方法が種々提案されている[58),60)]．

また，Esprit の DEMON (DEsign Methods based On Nets) プロジェクトでは，分散処理システムの形式的仕様記述と解析を主眼として，ペトリネットと CCS や OBJ などとの統合モデルが研究開発されている[7)]．

9. 今後の課題

従来，ペトリネット研究の重点が理論面に置かれ，実際のシステム開発への応用があまり進まなかった．しかし，1980 年代に SPN や HPN などの開発と軌を一つにして分散処理システムの開発や性能評価など，実システム開発への適用が進んだ．この結果，産業界におけるペトリネットによる開発事例が報告されるようになった．しかし，その方法は対象とするアプリケーション領域に依存しており，開発方法が十分体系化されるまでには到っていない．今後，理論的研究とともに，実開発における経験を蓄積，体系化し大規模分散処理システムなどへの適用を進める必要がある．

10. ま と め

ペトリネットなどのネット理論に基づくシステム開発方法は 1980 年代に飛躍的に発展した．プレース・トランジション・ネットは依然ペトリネットの基本であるが，SPN や HPN などの高水準な記述を可能とするモデルが開発されている．

一方，プロセッサ技術などの発展による分散処理システムの普及に対し，ソフトウェア開発技術が立ち遅れている．理論の裏打ちや定量的な評価に基づく体系的な分散処理システムの開発方法とその支援環境の開発が急務である．ペトリネットなどのネット理論に基づく開発方法は，このような開発を支援する優れた方法と考えられる．今後の発展と適用の推進を期待したい．

参 考 文 献

1) Akatsu, M. et al.: Verification of Error Recovery Specification for Distributed Data by Using Colored Petri Nets, IEICE Trans. on Fundamentals, Vol. E 74-A, No. 10, pp. 3159-3167 (1991).
2) Ammar, H. H. et al.: Hierarchical Models for Systems Reliability, Maintenability, and Availability, IEEE Trans. Circuits and Systems, Vol. CAS-34, No. 6, pp. 629-637 (1987).
3) Ammar, H. H. et al.: Performability Analysis of Parallel and Distributed Algorithms, Proc. IEEE PNPM '89, Kyoto, pp. 240-248 (1989).
4) Aoyama, M. and Chang, C. K.: A Petri Net Based Platform for Developping Communication Software Systems, IEICE Trans. Fundamentals, Vol. E 75-A, No. 10, pp. 1348-1359 (1992).
5) 青山幹雄：分散処理システムとネット理論，第5回電子情報通信学会回路とシステム軽井沢ワークショップ・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. 25-39 (1992).
6) 青山幹雄：ペトリネットによるシステム開発の方法，ソフトウェア設計者のためのペトリネット道場，日本ソフトウェア科学会 (1993).
7) Best, E.: Overview of the Results of the Esprit Basic Research Action DEMON: DEsign Methods based On Nets, Proc. IEEE PNPM '91,

Melbourne, pp. 224-235 (1991).
8) Bochmann, G. v.: Finite Description of Communication Protocols, Computer Networks, Vol. 2, No. 4, pp. 361-367 (1978).
9) Brand, D. and Zafiropulo, P.: On Communicating Finite State Machines, J. ACM, Vol. 30, No. 2, pp. 323-342 (1983).
10) Brofferio, S. C.: A Petri Net Control Unit for High-Speed Modular Signal Processors, IEEE Trans. Comm., Vol. COM-35, No. 6, pp. 577-583 (1987).
11) Bruno, G. and Balsamo, A.: Petri Net-Based Object-Oriented Modelling of Distributed Systems, Proc. ACM OOPSLA '86, pp. 284-293 (1986).
12) Chang, C. K. et al.: A New Design Approach of Real-Time Distributed Software Systems, Proc. IEEE COMPSAC '87, Tokyo, pp. 474-479 (1987).
13) Chen, S.-M. et al.: Knowledge Representation Using Fuzzy Petri Nets, IEEE Trans. Knowledge and Data Eng., Vol. 2, No. 3, pp. 311-319 (1990).
14) Chikofsky, E. et al.: Reverse Engineering and Design Recovery, IEEE Software, Vol. 7, No. 1, pp. 13-17 (1990).
15) Chiola, G.: A Software Package for the Analysis of Generalized Stochastic Petri Net Models, Proc. IEEE Int'l Workshop on Timed Petri Nets, Torino, pp. 136-143 (1985).
16) Couvillion, J. A. et al.: Performability Modeling with UltraSAN, IEEE Software, Vol. 8, No. 5, pp. 69-80 (1991).
17) Davis, A. M.: Software Requirements: Analysis and Specification, Prentice-Hall (1990).
18) Dugan, J. B. and Ciardo, G.: Stochastic Petri Net Analysis of a Replicated File System, IEEE Trans. Software Eng., Vo. 15, No. 4, pp. 394-401 (1989).
19) Embley, D. E. et al.: Object-Oriented Systems Analysis: A Model Driven Approach, Yourdon Press, Englewood Cliffs (1992).
20) Filman, R. E. and Friedman, D. P.: Coordinated Computing: Tools and Techniques for Distributed Software, McGraw-Hill, New York (1984).
21) Genrich, H. J.: Predicate/Transition Nets, Advances in Petri Nets 1986: Part I, Lecture Notes in Computer Science, Vol. 254, Springer-Verlag, Berlin, pp. 207-247 (1987).
22) Granda, M. et al.: Performance Evaluation of Parallel Systems by Using Unbounded Generalized Stochastic Petri Nets, IEEE Trans. Software Eng., Vol. 18, No. 1, pp. 55-71 (1992).
23) Harel, D. et al.: STATEMATE: A Working Environment for the Development of Complex Reactive Systems, IEEE Trans. Software Eng., Vol. 16, No. 4, pp. 403-414 (1990).
24) Harel, D.: Biting the Silver Bullet, IEEE Computer, Vol. 25, No. 1, pp. 8-20 (1992).
25) 長谷川晴朗他：ペトリネットを利用した並列処理プログラムの解析，第3回電子情報通信学会回路とシステム軽井沢ワークショップ論文集，pp. 199-206 (1990).
26) 平井健治他：分散制御システムのデバッグ手法：要求仕様を用いたイベントヒストリの検査，情報処理学会論文誌，Vol. 33, No. 4, pp. 491-500 (1992).
27) 本位田真一他：時制論理とペトリネット，オペレーションズリサーチ，Vol. 32, No. 9, pp. 612-618 (1987).
28) Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use: Vol. 1, EATCS Monographs on Theor. Computer Sci., Springer-Verlag, Berlin (1992).
29) Kavi, K. M. et al.: Isomorphism Between Petri Nets and Dataflow Graphs, IEEE Trans. Software Eng., Vol. 13, No. 10, pp. 1127-1134 (1987).
30) Kröger, F.: Temporal Logic of Programs, EATCS Monographs on Theor. Computer Sci., Vol. 8, Springer-Verlag, Berlin (1987).
31) 熊谷貞俊：ペトリネットツール，計測と制御，Vol. 28, No. 9, pp. 770-774 (1989).
32) 熊谷貞俊：ネット理論とその応用—I：ペトリネットと並行システム記述，システム／制御／情報，Vol. 34, No. 10, pp. 591-595 (1990).
33) Leveson, N. G. and Stolzy, J. L.: Safety Analysis Using Petri Nets, IEEE Trans. Software Eng., Vol. 13, No. 3, pp. 386-397 (1987).
34) Mandrioli, D. et al.: Modeling the Ada Tasking System by Petri Nets, J. of Computer Languages, Vol. 10, No. 1, pp. 43-61 (1985).
35) Manna, Z. and Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems Specification, Springer-Verlag, Berlin (1992).
36) Marsan, M. A. et al.: A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems, ACM Trans. on Computer Systems, Vol. 2, No. 1, pp. 93-122 (1984).
37) Marsan, M. A. et al.: Performance Models of Multiprocessor Systems, MIT Press, Cambridge (1986).
38) Marsan, M. A. et al.: On the Construction of Abstract GSPNs: An Exercise in Modeling, Proc. IEEE PNPM '91, Melbourne, pp. 2-17 (1991).
39) McDowell, C. E. and Helmbold, D. P.: Debugging Concurrent Programs, ACM Computing Surveys, Vol. 21, No. 4, pp. 593-622 (1989).
40) Mikkilineni, K. P. et al.: Petri-Net-Based Modeling and Evaluation of Pipelined Processing of Concurrent Database Queries, IEEE Trans. Software Eng., Vol. 14, No. 11, pp. 1656-1667 (1988).
41) Molloy, M. K.: Petri Net Modeling: The Past, the Present, and the Future, Proc. IEEE PNPM '89, Kyoto, pp. 2-9 (1989).
42) 村越英樹他：ペトリネットに基づくマルチマイク

ロプロセッサの制御方式，電子情報通信学会論文誌，Vol. J 70-D, No. 7, pp. 1285-1293 (1987).
43) Murata, T. and Zhang, D.: A Predicate-Transition Net Model for Parallel Interpretation of Logic Programs, IEEE Trans. Software Eng. Vol. 14, No. 4, pp. 481-497 (1988).
44) 村田忠夫：ペトリネットの解析と応用，近代科学社，東京 (1992).
45) Ozsu, M. T.: Modeling and Analysis of Distributed Database Concurrency Control Algorithms Using an Extended Petri Net Formalism, IEEE Trans. Software Eng., Vol. 11, No. 10, pp. 1225-1240 (1985).
46) Peterka, G. and Murata, T.: Proof Procedure and Answer Extraction in Petri Net Model of Logic Programs, IEEE Trans. Software Eng., Vol. 15, No. 2, pp. 209-217 (1989).
47) Pinci, V. O. and Shapiro, R. M.: An Integrated Software Development Methodology Based on Hierarchical Colored Petri Nets, Advances in Petri Nets 1991, Lecture Notes in Computer Science, Vol. 524, Springer-Verlag, Berlin, pp. 227-252 (1991).
48) Plunnecke, H.: Bibliography of Petri Nets 1990, Advances in Petri Nets 1991, Lecture Notes in Computer Science, Vol. 524, Springer-Verlag, Berlin, pp. 317-572 (1991).
49) Reisig, W.: A Primer in Petri Net Design, Springer-Verlag, Berlin (1992).
50) 離散事象システム研究専門委員会（編）：ペトリネットとその応用，計測自動制御学会 (1992).
51) Rumbaugh, J. et al.: Object-Oriented Modeling and Design, Prentice-Hall, Englewood Cliffs (1991).
52) Shatz, S. M. and Cheng, W. K.: A Petri Net Framework for Automated Static Analysis of Ada Tasking Behavior, J. Systems and Software, Vol. 8, pp. 343-359 (1988).
53) Shatz, S. M. and Wang, J.-P.: Distributed-Software Engineering, IEEE Computer Society Press, Washington D. C. (1989).
54) 椎塚久雄：実例ペトリネット，コロナ社，東京 (1992).
55) 白鳥則郎：試験検証技術，情報処理, Vol. 28, No. 4, pp. 403-410 (1987).
56) Someya, H. et al.: Performance Evaluation of Job Operation Flows in Computer Systems by Timed Petri Nets, Proc. IEEE PNPM '89, Kyoto, pp. 104-111 (1989).
57) 菅沢喜男，瀬谷浩一郎：取り替えを考慮した信頼性システムのペトリネット表現によるモデル化の信頼度解析，電子情報通信学会論文誌，Vol. J 69-A, No. 11, pp. 1301-1309 (1986).
58) Suzuki, T. et al.: A Protocol Modeling and Verification Approach Based on a Specification Language and Petri Nets, IEEE Trans. Software Eng., Vol. 16, No. 5, pp. 523-536 (1990).
59) Suzuki, I.: Formal Analysis of the Alternating Bit Protocol by Temporal Petri Nets, IEEE Trans. Software Eng., Vol. 16, No. 11, pp. 1273-1281 (1990).
60) Uchihira, N. and Honiden, S.: Verification and Synthesis of Concurrent Programs Using Petri Nets and Temporal Logic, 第3回電子情報通信学会回路とシステム軽井沢ワークショップ論文集, pp. 183-190 (1990).
61) Voss, K.: Using Predicate/Transition-Nets to Model and Analyze Distributed Database Systems, IEEE Trans. Software Eng., Vol. 6, No. 6, pp. 539-544 (1980).
62) Watanabe, T. et al.: A Petri Net-Based Algorithm for the Satisfactability Problem in the Horn Clause Propositional Logic, 第3回電子情報通信学会回路とシステム軽井沢ワークショップ論文集, pp. 191-198 (1990).
63) 渡辺敏正他：ホーン節命題論理の証明可能性とその関連問題，第4回電子情報通信学会回路とシステム軽井沢ワークショップ論文集, pp. 402-407 (1991).
64) Xu Z., and Vel, O. de: Petri Net Modelling of Occam Programs for Detecting Indeterminancy, Non-Termination and Deadlock Anomalies, Proc. IEEE PNPM '91, Melbourne, pp. 116-124 (1991).
65) Yau, S. S. and Caglayan, M. T.: Distributed Software System Design Representation Using Modified Petri Nets, IEEE Trans, Software Eng., Vol. 9, No. 6, pp. 733-745 (1983).

（平成5年1月18日受付）

青山 幹雄（正会員）

1980年岡山大学大学院工学研究科修士課程修了．同年富士通(株)入社．現在，同社ビジネス通信事業本部にて分散処理ソフトウェアシステムの開発手法，開発支援環境，ソフトウェア開発プロセスの管理などの開発と適用に従事．1986～88年米国イリノイ大学客員研究員．IEEE Software の Editor, IEEE COMPSAC などのプログラム委員．電子情報通信学会，ソフトウェア科学会，IEEE, ACM 各会員．