ソフトウェア工学　52－13
（1987・2・14）

並行処理ソフトウェアシステムの設計向き プロトタイピング手法とそのツール

田村恭久，伊藤　潔

上智大学　理工学部

並行処理ソフトウェアシステムの設計段階に適用されるソフトウェアプロトタイピング手法とそのツールについて述べる．並行処理ソフトウェアシステムは多数の並行処理モジュールから成り，それらが並行して多数のトランザクションを処理するシステムである．機能面での設計のためには，トランザクションの種別・ルート作り，排他的なアクセスを必要とする資源や構成要素の識別，内部でのアルゴリズム・実行手順の設計が必要である．また，資源の利用度合いや実行時間などを見積り・評価する性能面の設計作業が必要である．以上の並行処理ソフトウェアシステムの設計のために「ステップワイズプロトタイピング」手法を考案し，パソコン上で並行処理ソフトウェアシステムのプロトタイプを稼働させ，機能面での振舞いを視認し，性能評価データを収集させるために，Prolog言語処理系を活用したP-Flots(Prolog based FLOw and Task Simulator)を開発した．

Prototyping Method and Tool for Design Process of Concurrent Software Systems

Yasuhisa TAMURA , Kiyoshi ITOH

Faculty of Science and Technology , Sophia University
7-1 Kioi-cho Chiyoda-ku Tokyo 102 Japan

A prototyping method and a tool are proposed for designing concurrent software systems. In concurrent software design process, the software designer is required to design the systems from both their function and performance aspects. As the design proceeds, the designer should recognize "tasks" each of which must be used serially and exclusively. We propose a "stepwise prototyping" methodology and a corresponding tool P-Flots (Prolog based FLOw and Task Simulator) for designing the systems. P-Flots executes the prototypes and visualizes their dynamic behavior and performance. The designer is capable of designing the systems in stepwise from their function and performance aspects with use of our methodology and P-Flots.

1　まえがき

並行処理ソフトウェアシステムの設計段階に適用されるソフトウェアプロトタイピング手法とそのツールを述べる．

ソフトウェアプロトタイプとは，ソフトウェアシステムの元になる型であり，その実現方式は開発対象のソフトウェアシステムとは異なっていてもよいが，開発対象のソフトウェアシステムの稼働する実環境あるいはその模擬環境で，初めから動くものである．

ソフトウェアシステムの開発に用いられるプロトタイプの性質として，幾つかの文献（たとえば文献[10]）を総合すると表1の4点が挙げられる．

図1に示す通り，ソフトウェアプロトタイピングとは，対象となるソフトウェアシステムの開発の際に，表1の4つの性質をもったプロトタイプを作成することにより，対象のソフトウェアシステムの稼働（模擬）環境の上でプロトタイプを稼働させながら，それを迅速にかつ徐々に進化・精密化しながら，ユーザや顧客の要求や設計者の実現方法を明確化する開発手法である．プロトタイプが要求分析段階に適用される場合には，稼働状態のプロトタイプの振舞いを見て，要求のあいまいさや誤りが検出され，要求が引き出され確認される．設計段階に適用される場合には，設計者は要求分析段階で決定された要求項目を満たしつつ，プロトタイプの振舞いを確認しながら，対象システムの中で採用されるアルゴリズムやデータ構造を選択・設計する．対象システムによっては，プロトタイピング手法を用いた場合，厳密に開発フェーズを分ける必要はない場合もあり，要求分析のサイクルと設計のサイクルを任意に切り換えながら回ってプロトタイプを作成する．

筆者らは，並行処理ソフトウェアシステムの設計段階に対してプロトタイピングの適用効果があると考えている．並行処理ソフトウェアシステムは，通常の逐次処理ソフトウェアシステムにはない複雑さをもった多数の並行処理モジュールから成り，それらが並行して多数のトランザクションを処理するシステムである．このような並行処理ソフトウェアシステムの設計では，システムの内部構造やアルゴリズムの選択・決定，およびその妥当性の検討という機能面での設計の作業がある．このためにはまず多数のトランザクションを種別し，システム内でのそれらのルート作りを行う．このルートを徐々に詳細化していく過程の中で，排他的なアクセスを必要とするシステム内の資源や構成要素を正しく識別し，それらをさらに詳細化しながら，互いに並行的に稼働する並行処理モジュール群を定め，個々のモジュールの内部で採用されるアルゴリズムや実行手順を設計していく．

このような並行処理モジュール群の識別と詳細化の過程と共に，それらの性能面の設計の作業が並行処理ソフトウェアシステムの設計において特に必要である．このためには，機能面での設計の過程の中で，詳細化に従って徐々に明確となる資源の利用度合いや並行処理モジュールの実行時間などの見積もりを正しく導入することが必要である．

筆者らは以上の要件を満たすために，並行処理ソフトウェアシステムの設計のためのプロトタイピング手法として「ステップワイズプロトタイピング」を考案した．筆者らはこれまで汎用大型機上で，並行処理ソフトウェアシステムの設計のために構造化プログラミングに基づく枠組みをもつ設計言語とその処理系Duvis (DUal-View Integration Simulator)[5]を開発し，さらにこれを一般的に流布しているGPSSを用いてG-Flots (GPSS based FLOw and Task Simulator)[6][11]を開発してきた．これらの2つのシステムの開発の経験の下に，より精密に並行処理モジュール群を識別し設計できる手順をもつステップワイズプロトタイピングを考案し，さらにパソコン上で並行処理ソフトウェアシステムのプロトタイプを稼働させ，機能面での振舞いの評価を視認でき性能評価データの収集を自動化させるために，Prolog言語とその処理系を活用したP-Flots(Prolog based FLOw and Task Simulator)を開発した．

並行処理ソフトウェアシステムの設計方法論としてCamposの提案したSARA(System ARchitecture Apprentice)[1]がよく知られている．SARAはどちらかというとソフトウェアの初期設計フェーズ以後の階層的な設計と評価を行うことに対して，ステップワイズプロトタイピングは設計フェーズ全般を組織化しようとする．並行処理ソフトウェアシステム向きのプロトタイピング手法としてZaveの提案したPAISley[12]がある．これは並行プロセス自体と並行プロセス間の関係を関数型プログラミング言語で記述しシミュレーションを行うもので，並行プロセスとなるものを設計の中で識別する手法であるステップワイズプロトタイピングとは異なる．

Prolog言語を用いたプロトタイピング手法としては，状態遷移図シミュレータ[3][8]があるが，それらは状態遷移図自体を作成する設計手順をもたず，Prolog言語とその処理系を活用したシミュレータの開発にとどまっている．

以下の節では，ステップワイズプロトタイピングについてのより詳細な議論と，プロトタイピングツールとしてのP-Flotsの特徴を順次述べる．

表1　ソフトウェアプロトタイプのもつべき性質

稼働性	プロトタイプは，その内部での実現方式（あるいは実行方式）と異なっていても良いが，最初から稼働可能であること．その稼働の際に対象システムの機能や性能を調べることができること．
環境導入性	プロトタイプは，対象システムの稼働する実環境（あるいはその模擬環境）上で稼働可能であること．
作成の迅速性	プロトタイプには，対象システムのもつべき機能や性能項目のうち，着目した項目の機能や性能を迅速に装備可能であること．
進化性	プロトタイプは，全面的な版の改変ではなく徐々に精練・精密化して版を進化可能であること．時には，プロトタイプの最終版を実際の対象システムとできる場合もあること．

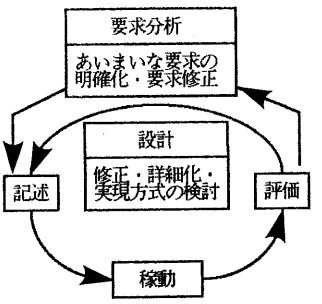

図1　ソフトウェアプロトタイピングサイクル

2　並行処理ソフトウェアシステムの設計

ここで対象とするのはオンラインソフトウェアシステムやデータベース問い合わせシステムなどの並行処理ソフトウェアシステムである．これらは，トランザクション・メッセージ・検索質問など（「処理要求」とよぶ）によって駆動され，複数のモジュール（タスクあるいはプロセス，関数，プロシジュア等）によって協同しながら処理要求を処理するシステムととらえることができる．

並行処理ソフトウェアシステムでは，複数の処理要求がシステムの外で発生し，各々が並行処理ソフトウェアシステム内部の指定されたルートに従って，その上を流れながら処理を受ける．複数の処理要求は，互いに排他的な使用を必要とする資源を使ったり，処理要求に対するシステムからの排他的な処理を受けなければならない場合がある．この場合，排他的なアクセスを制御する機構によって排他制御が実現され，その部分を通る処理要求は，アクセス権を得るために待ち行列に並ばなければならない．並行処理ソフトウェアシステムの設計では，このような排他的なアクセスを必要とする部分「逐次的再利用可能部分」を浮かび上がらせ，排他的なアクセスを必要としない部分「再入可能部分」と明確に分離していく作業が重要である．逐次的再利用可能部分は，最終的なソフトウェアシステムの中でタスクとして実現されるのが普通であろう．

並行処理ソフトウェアシステムの機能面の設計は，処理要求を正しく処理するためのルート作りと，そのルートの中で，排他的なアクセスを必要とする逐次的再利用可能部分と，排他的なアクセスを必要としない再入可能部分との分離・明確化を主眼とするプロセスである．

概要のルートを作成し，それをより詳細にしていくプロセスの中で，設計の詳細化に従って視点が徐々に変化する，すなわち，処理要求の動きの記述が主体である視点（「フロー指向パラダイム」とよぶ）から，処理要求を逐次的に排他的に処理する逐次的再利用可能部分，いわゆるタスクの機能の記述が主体である視点（「タスク指向パラダイム」とよぶ）へ徐々に変化する過程が，存在すると考えられる．パラダイムがフロー指向からタスク指向へ徐々に変化する過程の中で，ソフトウェアシステムの実現方法が徐々に明らかにされてゆく．実現方法を明らかにするとは，ルートをより詳細に決定すること，より詳細になった部分を更に分離・明確化すること，分離・明確化されたものの中で，タスクとなった部分が処理要求を正しく処理する手順やアルゴリズムを設計すること，および，タスク間で並行処理を制御する機能を使ってタスク間の動的な関係を設計することである．

並行処理ソフトウェアシステムの設計では，ここまで述べた機能面の設計と共に性能面の設計も考慮しなければならない．並行処理ソフトウェアシステムの性能設計は，そのシステムの動的な側面に大きく依存する．並行処理ソフトウェアシステムの性能設計とは，複数の処理要求とタスクをもち，それらが多数並行的に動作し，必要なところで互いに排他的な処理をする並行処理ソフトウェアシステムに対して，そのシステムに対して課せられた性能要求項目が設計上実現可能か否かを調べることである．

並行処理ソフトウェアシステムの動的な側面を構成する要素として，システム中の処理要求やタスクの個数，処理要求やタスクの動的な関係，処理要求やタスクが使用する資源の量と時間などがある．これらが明らかになって初めて，処理要求に対する応答時間や，そのシステム内部での待ち時間や待ち行列長などの，性能面の要求事項の実現可能性が現実に明らかになってくる．要求分析段階では，性能面の要求事項はシステムへの制約事項として要求仕様に現れている場合が多く，設計段階で，この性能面での要求項目を満たす設計が必要である．それを満たしているか否かは，処理要求やタスクが数え上げられ，それらの動的な関係が明らかにされる機能面での設計と同時に，並行処理ソフトウェアシステムでは明らかになる場合が多い．さらに，これらは設計中の並行処理ソフトウェアシステムを，それが実際に稼働する環境 ― 設計段階であるため，実際の環境であっても，その模擬環境であってもどちらでもよい ― で動かしてみなければわからない場合が多い．

以上の設計過程を図2に示す．ここでは性能要求項目の制約の下に機能要求項目を満たすように並行処理ソフトウェアシステムを詳細化しながら設計を進めていくことが示されている．

3　並行処理ソフトウェアシステムのステップワイズプロトタイピング

プロトタイプを使った設計は，記述→稼働→評価→詳細記述→…というサイクルで表わすことができる．並行処理ソフトウェアシステムのプロトタイプは，設計初期にはフロー指向パラダイムに基づく抽象度の高いレベルで記述され，設計が進展するにつれて，徐々に詳細化され，実現方法を意識した形に記述が変化していく必要がある．すなわち，パラダイムがフローの視点からタスクの視点に変化することに柔軟に対応しなければならない．

フロー指向の設計記述であるプロトタイプも，タスク指向を導入した設計記述であるプロトタイプも，実システムの稼働環境，あるいはその模擬的な稼働環境で稼働させることが必要である．

プロトタイプを稼働させながら，その機能のテストが必要となる．これはプロトタイプの振舞いの視認による処理要求のフローの正しさの確認である．これには並行処理ソフトウェアシステムの開発に

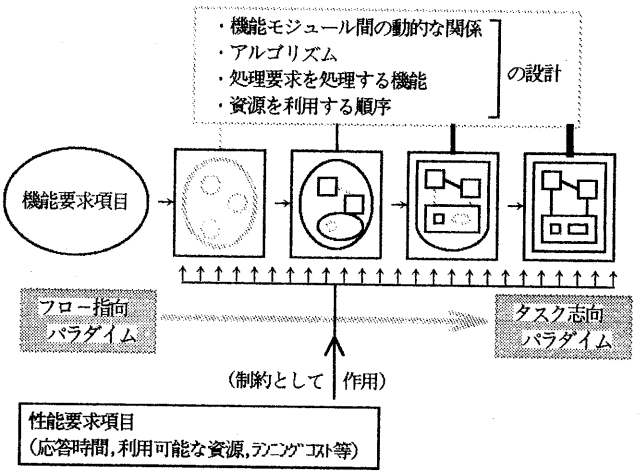

図2　並行処理ソフトウェアシステムの設計

必要不可欠なデッドロックの検出も含まれる．

さらにプロトタイプの性能評価も必要となる．これはシステム中の処理要求やタスク数の測定，処理要求やタスクの待ち行列長や要求する資源の量・時間の測定などについてなされる．

ステップワイズプロトタイピングに導入されているフロー指向パラダイムに基づく設計方法論は，段階的な分解と詳細化に基づくトップダウン的な過程である．フロー指向パラダイムの中に徐々にタスク指向パラダイムを取り入れていく設計方法論は，段階的な置き換えと詳細化の過程である．

図3にステップワイズプロトタイピングの手順を述べる．このプロトタイピングの過程の中で，処理要求の処理されるルートが明確化され，分解されていく．その中で処理の詳細な実現方法は明らかにされていないが，行なわなければならない処理の内容が明確となったものが浮かび上がってくる．これを総称して「機能モジュール」と呼ぶことにする．機能モジュールには，処理要求にとって排他的なアクセスを必要とする部分（「逐次的再利用可能機能モジュール」あるいは「タスク」）と，排他的なアクセスを必要としない部分（「再入可能機能モジュール」）とがある．

処理要求のフローの視点を中心とした，フロー指向パラダイムによる設計の前半は，図3(A)に示す通り，逐次的再利用可能機能モジュールと再入可能機能モジュールを処理要求のフローの記述のなかで，明確に分離して浮かび上がらせていくプロセスである．

このフロー指向パラダイムに基づく設計では，処理要求が機能モジュールを使うという立場の記述であるが，図3(B)の手順は，タスク指向パラダイムによる設計を混在させて，インプリメンテーションのための並行処理ソフトウェアシステムの骨格を浮かび上がらせる手順である．

プロトタイプに対する模擬的な実行において機能面でテストされる項目は図4(A)の通りである．性能評価の項目については機能モジュールや資源の利用度合に対して図4(B)の項目がある．また，処理要求の種類ごとに図4(C)の項目がある．

(1) 処理要求を数え上げ種別する．

(2) 設計対象の並行処理ソフトウェアシステム全体を，1つの機能モジュールとみなす．

(3) 逐次的再利用可能機能モジュールが全て数え上げられるまで，再入可能機能モジュールについては設計者が満足するまで，以下の手順を繰り返す．

(a) 任意の機能モジュールに着目し，その着目した機能モジュール内での種類の処理要求のフローを記述する．この過程の中で，より細分化した機能モジュールを数え上げ，それらの機能モジュールを処理要求が使うという形で記述する．機能モジュールが逐次的再利用可能機能モジュールか再入可能モジュールかによって，前者の場合，それを排他的に処理要求が使うことを明確に記述する．

(b) 処理要求が構成要素を使う際の所要時間および空間的な使用量の見積もりを与える．

(c) フロー指向のプロトタイプを動的に模擬的に稼働させる．この際に機能のテストを行い性能の評価データを調べる．

(d) 機能のテスト結果を検討して不都合があれば(a)に戻ってフロー指向の記述を正す．

(e) 性能の評価データを検討して不都合があれば(b)に戻って所要時間見積もりあるいは空間的な使用量を再検討する．

(A) フロー指向パラダイムによるプロトタイピング手順

設計者が満足するまで逐次的再利用可能モジュールについて，
以下の手順を繰り返す．

(a) 逐次的再利用可能機能モジュールの記述を，処理要求をシステム内の資源（たとえば，ファイル，データベース，様々な機器など）を使って処理するという立場で書き直す．この詳細の程度は任意である．

(b) 逐次的再利用可能モジュールの実行時間および空間的な使用量の見積もりを資源の利用度合などを考慮しながら与える．

(c) フロー指向とタスク指向の共存したフロトタイプを動的に模擬的に稼働させる．この際に機能のテストを行い性能の評価データを調べる．

(d) 機能のテスト結果を検討して不都合があれば(a)に戻って記述を再検討する．

(e) 性能の評価結果を検討して不満足であれば(b)に戻って実行時間あるいは空間的な使用量の見積もりを再検討する．

(B) タスク指向混在によるプロトタイピング手順

図3　ステップワイズプロトタイピング手順

4　Prologによるプロトタイピングシステムの実現

並行処理ソフトウェアシステムのためのステップワイズプロトタイピングを，パソコン上で稼働するProlog言語とその処理系を活用して実現したP-Flots (Prolog based FLOw and Task Simulator) を述べる．P-Flotsの実現のために使われるPrologの言語機能は，市販のパソコンベースのProlog処理系（たとえば文献[4][7]）間でほぼ共通な言語機能である．

ステップワイズプロトタイピングを実現するためには，フロー指向パラダイムやタスク指向パラダイムを実現す

不正なフロー	処理要求が意図しない所を走っている
不正な状態	機能モジュールや資源が意図しない状態になった
デッドロック	処理要求のフローが機能モジュールや資源の取り合いで互いに止まってしまった
不正な使用	処理要求が使っていない機能モジュールや資源を解放しようとした

(A) 機能評価項目

図4　プロトタイプの評価項目（続く）

るための稼働系，機能のテストを可能とする対話型稼働環境および性能評価情報の収集機能の実現と，ステップワイズプロトタイピングの手順に適合したプロトタイピング言語の実現が必要である．

4.1 フロー指向パラダイムの実現

並行処理ソフトウェアシステムは，複数の処理要求が同時に非決定的な状態遷移を起こすモデルとしてとらえられる．Prolog処理系を活用すると，1つの処理要求が並行処理ソフトウェアシステム内に存在することと，1つの事実がProlog処理系のもつ作業領域に存在することを1対1に対応できる．1つの処理要求が複数の属性をもつことは，1つの事実が複数の引数をもつことと1対1に対応できる．処理要求が並行処理ソフトウェアシステムの中で状態遷移することと，事実のもつ引数がP-Flotsの処理系（これは規則の形でProlog言語により実現されている）に従って更新されることも1対1に対応できる．さらに複数の処理要求がシステム内に存在することは，Prolog処理系の作業領域の中に複数の事実が同時に存在することに対応できる．

処理要求が競合して使用する資源も，作業領域内に事実として蓄積される．資源は状態（freeかbusyのいずれか）を引数としてもつ事実として表され，もしfreeならば，1つの処理要求の状態が「待ち」から「使用開始」に更新され，これに応じて資源の状態はfreeからbusyに更新される．その処理要求の状態が「使用終了」に更新されることに従って，資源の状態がfreeに更新される．これらの更新もP-Flotsの処理系によって管理される．

P-Flotsでは，稼働するプロトタイプ内部の時刻の管理も事実の更新と対応させ，ある時刻で動き得る処理要求がなくなると，時刻に対応する事実を更新する．

処理要求は資源の競合などによって待ち行列を形成するため，同じ時刻での処理要求間の順序関係を規定と管理を行う必要がある．

Prolog処理系では，非決定的なユニフィケーションを行なう場合，ユニフィケーション可能な事実の間に順序関係を便宜的にもたせ，その順番に従ってユニフィケーションを行なう．P-Flotsでは，このProlog処理系がもつ事実間の順序付けを利用し，この待ち行列管理の問題をProlog処理系に任せている．Prologでは事実を更新する際に，その事実と同じ述語名をもつ他の事実との間の順序関係を指定できる（retract・assert述語などの活用）．これにより，待ち行列が発生する場所などで，処理要求間の順序関係を指定することが可能となる．この順序付けにより，待ち行列を形成している処理要求のどれを最初に扱うか，また待ちの状態に遷移した処理要求が行列のどこに並ぶか，ということが簡単に実現できる．

以上の同時に動き得る処理要求を順序立てて動かす機構の考え方は，トランザクション指向のシミュレータであるGPSS[9]の基本的な考え方と同じである．

以上により並行処理ソフトウェアシステムの多数の処理要求のフローシミュレーションがPrologによって実現可能である．

P-Flotsを使う並行処理ソフトウェアシステムの設計者は，個々の処理要求のシステム内での流れを，事実の羅列の形で表し，P-Flotsの処理系に与えて，プロトタイプの記述・稼働・評価のサイクルを進めることができる．

4.2 タスク指向パラダイムの実現

設計が進むにつれて，タスク指向パラダイムによる処理の記述を行い，これを組み込んで稼働，評価しながら詳細化してゆく．タスク指向パラダイムは，Prolog規則の呼出しを通じて可能である．その規則の中で

(1)Prolog処理系の組込み述語“system”による実行可能プログラムファイルの呼出し，あるいは

(2)通信ポートを介した他のパソコン上での実行可能プログラムファイルの呼出し

が可能である．実行可能プログラムは通常の手続き型言語によって書かれたものであり，これはコンパイル・リンクを経た実行可能形式で起動されるタスクである．このように手続き型言語で書かれたタスクを直接呼び出すことも可能であるが，P-Flotsを使う設計者の便宜を図るように，いくつかのP-Flots組込み述語 — たとえば，ファイルアクセスや通信手順を標準化した述語など — が用意され，手続き型言語で書かれたタスクの呼び出しの前後で，それらを利用できるようにしている．また，これとは対照的に，Prologを書き慣れた設計者は，全てProlog組み込み述語を使ってProlog規則の形でタスクを記述することも可能である．

4.3 対話型稼働環境の実現

並行処理ソフトウェアシステム向きプロトタイピングツールをPrologで実現することの利点は，いわゆるオンラインシミュレーション機能の実現にある．Prolog処理系では，ある時点での（更新途中の）事実を観察することや，任意の時点で実行の中断・再開が容易

使用総数	個々の機能モジュールや資源を使った処理要求の総個数
平均使用量	個々の機能モジュールや資源を単位時間当たり何個の処理要求が使っていたかを示す
平均使用率	平均使用量を機能モジュールや資源の容量（最大いくつの処理要求がこれらを使えるか示す）で割ったもの
平均使用時間	使用時間の個数平均
平均待ち行列長	待ち行列に並ぶ処理要求の平均個数
平均待ち時間	処理要求が待ち行列に並んでいる時間の平均値

(B)利用度合に関する性能評価項目

生成個数	何個生成したか
終了個数	何個終了したか
経過時間	生成してから終了するまで，あるいは指定区域を通った時の所要時間の最大・最小・平均

(C)処理要求に関する性能評価項目

図4　プロトタイプの評価項目

である．P-Flotsでは，Prolog処理系のこの性質を利用することにより，各時点でのプロトタイプの稼働状態のスナップショットを保存することができ，任意の時点でのプロトタイプの稼働を中断したり，任意の時点から再開（ロールバック）させることが可能である．

また，各時刻での各々の処理要求の更新状態を逐一出力できるので，これから待ち行列の形成の視認およびデッドロックの発生などの図4(A)で示した機能評価項目の視認が可能となる．

4.4 性能情報収集

P-Flotsには，図4(B)(C)に示した基本的な統計量を自動的に収集する機能がある．これは，事実の形で更新し保存することによって，シミュレーション中断時，あるいは終了時に出力できる．

4.5 P-Flots言語構成要素

P-Flotsのフロー指向での機能を図5に示す．

3節で述べたステップワイズプロトタイピングの手順に従ってプロトタイピングを行うためには，機能モジュール毎に，その中を動く処理要求について，その処理要求が発生してから消滅するまでに，ステップワイズプロトタイピングの手順に従ったある設計段階で，明確に浮かび上がった機能モジュールや資源を順次的に使用する際に起きる事象の発生の系列を，事実の系列で書き，また，システム内の各種資源の初期状態をアサートする事実の集まりを与えることによって行なう．

[時間経過，<int>]	<int>時間sleepする
[時間経過，[<int1>，<int2>]]	一様分布でsleepする
[使用開始，<資源名>]	排他使用開始
[使用終了，<資源名>]	排他使用終了
[ﾒｯｾｰｼﾞ送信，[<ﾒｲﾙﾎﾞｯｸｽ名>，<ﾒｯｾｰｼﾞ内容>]]	ﾒｯｾｰｼﾞ送信
[ﾒｯｾｰｼﾞ受信，[<ﾒｲﾙﾎﾞｯｸｽ名>，<ﾒｯｾｰｼﾞ内容>]]	ﾒｯｾｰｼﾞ受信
[実行，<機能モジュール名>]	機能モジュールの実行
[実行，<規則名>]	Prolog規則の起動
[if，条件]	条件による分岐
[else]	条件 if が失敗の時の分岐
[while，条件]	条件成立の間の繰返し
[repeat]	繰返し(until で条件設定)
[until，条件]	repeat の条件設定
[select]	成立している節の実行

図5　P-Flotsのフロー指向での機能

5 例題

設計例を示す．甲と乙という2種類の処理要求が存在し，その両者はファイル使用と後処理が必要であることが知られていると仮定する．P-Flotsを用いる設計者は，設計の初期段階で，この仕様を，処理要求の種類毎にエントリモジュールを設け，処理要求は各々のエントリモジュール内で「ファイル更新」と「後処理」を使用する，という形で記述する（図6(A)）．フロー指向の設計を進めていくにつれて，ファイル使用に先立って前処理が必要であること，ファイル使用の前と後に特定のルーチンが必要であること，ファイルアクセスは排他的な実行が必要であること，の3つが明らかになったとする（図6(B)）．この段階における設計者のP-Flots記述を図7に示す．ここで，甲と乙の発生時間間隔が10±2単位時間の一様分布と見積もられ，「ファイル更新」には4単位時間要すると見積もられている．

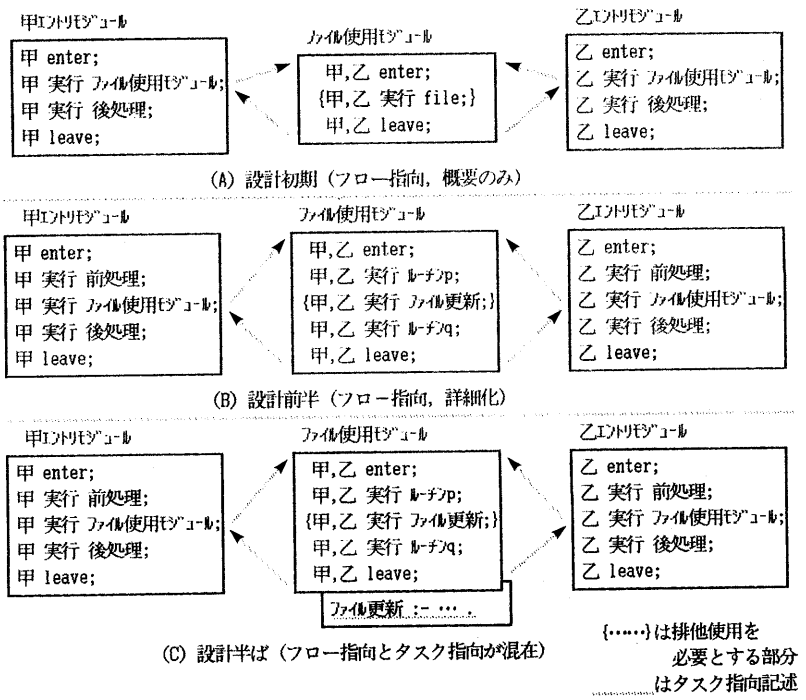

図6　システムの設計例

ここまで設計が進展した段階で，設計過程の中で浮かび上がった逐次的再利用可能機能モジュールをタスク指向で書き直すことを考える．ここでは，「ファイル更新」が逐次的再利用可能機能モジュールである．この「ファイル更新」の利用の前後で排他的使用の開始と終了が明示されている．図6(C)と図7の「ファイル更新」について，Prolog規則を呼び出し，別のパソコン上で記述されたタスクを呼出す記述が図8である．この記述では，設計者が書いた「更新規約」述語によってレコードの更新法が与えられる．別のパソコ

ン上のタスク呼出しのために，P-Flots組込み述語「通信ルーチン」により，呼出しのパラメータ（タスク名がupdate，そのパラメータがtype，key，typeはレコードの更新法，keyは更新するレコードのキィ）を送り，P-Flots組込み述語「レコード更新」によって与えられたキィのレコードを更新する．この結果，"update type key"（type，keyは整数）という文字列が通信ポートに送られ，受け手のパソコンでは，この文字列をコマンドとして解釈し，タスク update を起動することになる．受け手のパソコン上でC言語で記述されたタスクを図9に示す．

プロトタイプの稼働の終了時に出力された，ファイル更新モジュールに関する統計結果を，図10に示す．稼働途中のプロトタイプの稼働のスナップショットとファイル更新の様子を図11に示す．

6 むすび

P-Flotsにより開発されたプロトタイプが表1で示した性質を満たしているかどうかを検討する．

稼働性，作成の迅速性，進化性について．処理要求のフローのためにプリミティブな記述が用意されており，記述されたプロトタイプは稼働可能である．また，フローをより詳細な記述に書き改めることやあるいはタスク指向パラダイムを導入することは，P-Flotsのステップワイズプロトタイピング手順と言語構造に従えば比較的簡単である．しかし，プロトタイプの作成や修正をより迅速に進めるためには，より効果的なプロトタイプの編集系が必要であると考えられる．

環境導入性について．フロー指向では，実際のシステムの資源を模擬し，その利用時間を与えて稼働可能であり，その中で機能のテストを行いながら，資源の使用状況が把握できる．また，タスク指向の考え方では，実際のシステムの資源を使った処理を手続き型言語で記述し，これをフローの記述の中から呼出すことによって，実環境を組み込んで，より詳細な機能のテストと資源の使用状況が可能となる．

述語に並行性を導入し，それらの述語間のスケジュール機能，時刻管理機能および述語の中で用いられる資源の管理機能を導入したT-Prolog[2]とよぶシミュレーション言語とその処理系がある．これを用いればP-Flotsのインプリメンテーションや実行の効率が向上する可能性もあるが，P-Flotsの現バージョンはパソコン上で現段階で流布している通常のProlog言語を使用した．

P-Flotsは，結合されたIBM5540およびIBM5560（MS-DOS）上でインプリメントされ，P-Flots本体は，全体でProlog-Kaba言語で約760行，そのうち処理要求の状態遷移に関する部分が約340行となっている．IBM5540はフロー指向の設計作業に，IBM5560はタスク指向の設計作業に使われている．

今後の課題として，P-Flotsの適用例を増やして，共通的に利用できるP-Flots組込み述語の数を増やしたいと考える．また，デッドロックなどのP-Flots処理系による自動検出を実現していきたいと考える．

参考文献

1) Campos,I.M., et al. : Concurrent Software System Design Supported by Sara at the Age of One, Proc. 3rd ICSE , pp.230- 242 (May 1978).
2) Futo,I. et al. : T-Prolog User Manual Version 4.2, SZKI, (March 1983).
3) 本位田真一,松本吉弘:リアルタイムシステムにおけるプロトタイピングの1手法，情報処理学会論文誌，Vol.26，No.5,pp.946-953 (September 1985).
4) Intermedia Access Corporation:Prolog-J, (March 1985).
5) K.Itoh et al.:Software Design Process: Chrysalis Stage under the Control of Designers, Journal of Information Processing, Vol.7, No.1, pp.5-14 (March 1984).
6) 伊藤 潔，田畑孝一:ソフトウェア設計におけるプロトタイピング，bit臨増，pp.166-179 (July 1984).
7) Kyoto Artificial Brain Associates:Prolog-KABA Reference Manual (May 1985).
8) Lee,S.: On executable models for Rule-Based Prototyping, Proc. 8th ICSE, pp.212-215 (August 1985).
9) 日電:ACOS-6離散型シミュレーション言語説明書,GPSS/V6 (1984).
10) SIGSOFT Software Enginerring Note, Vol.7, No.5 (December 1982).
11) Y.Tamura and K.Itoh:Software Prototyping Using Simulation Language, Proc. JSST Conference on Recent Advances in Simulation of Complex Systems, pp.41-51 (July 1986).
12) Zave,P.:An Overview of the PAISLey Project, ACM SEN 9-4, pp.12-19 (1984).

```
処理要求発生(甲,10,2,甲ｴﾝﾄﾘｰ).
機能要素(甲ｴﾝﾄﾘｰ,[甲],[実行,前処理]).
機能要素(甲ｴﾝﾄﾘｰ,[甲],[実行,ﾌｧｲﾙ更新]).
機能要素(甲ｴﾝﾄﾘｰ,[甲],[実行,後処理]).

処理要求発生(乙,10,2,乙ｴﾝﾄﾘｰ).
機能要素(乙ｴﾝﾄﾘｰ,[乙],[実行,前処理]).
機能要素(乙ｴﾝﾄﾘｰ,[乙],[実行,ﾌｧｲﾙ更新]).
機能要素(乙ｴﾝﾄﾘｰ,[乙],[実行,後処理]).

機能要素(ﾌｧｲﾙ更新,[_],[実行,ﾙｰﾁﾝp]).
機能要素(ﾌｧｲﾙ更新,[_],[使用開始,ﾌﾗｸﾞ]).
機能要素(ﾌｧｲﾙ更新,[_],[実行,ﾌｧｲﾙ更新]).
機能要素(ﾌｧｲﾙ更新,[_],[時間経過,4]).
機能要素(ﾌｧｲﾙ更新,[_],[使用終了,ﾌﾗｸﾞ]).
機能要素(ﾌｧｲﾙ更新,[_],[実行,ﾙｰﾁﾝq]).

機能要素(前処理,[_],[時間経過,2]).

機能要素(後処理,[_],[時間経過,3]).

資源状態(ﾌﾗｸﾞ,free).
時刻(0).
終了時刻(200).

ﾙｰﾁﾝp.
ﾙｰﾁﾝq.
ﾌｧｲﾙ更新.
```

図7 図6(B)のP-Flots記述

```
ﾌｧｲﾙ更新 :-
    s_random(2,Type),
    s_random(50,Key),
    通信ﾙｰﾁﾝ(update,Type,Key),
    ﾚｺｰﾄﾞ参照(Key,Cont),
    更新規約(Type,Key,Cont,NewCont),
    ﾚｺｰﾄﾞ更新(Key,Cont,NewCont).

更新規約(1,Key,Cont,NewCont) :-
    NewCont is Cont * 2.
更新規約(0,Key,Cont,NewCont) :-
    NewCont is Cont / 2.

/* "通信ﾙｰﾁﾝ","ﾚｺｰﾄﾞ参照","ﾚｺｰﾄﾞ更新"
    の3述語は, P-Flots組込み述語として
    登録されている. */
```

図8 呼出されるProlog規則およびタスク呼び出し

```
#include <stdio.h>
#include <stdlib.h>
FILE *fp,*fopen();
int mf[50][2];

main(argc,argv)
int argc;
char *argv[];
{   int type,key,i;

    fp=fopen("master","r");
    for (i=0;i<50;i++)
        fscanf(fp,"%d %d",&mf[i][0],
                &mf[i][1]);
    fclose(fp);
    type=atoi(argv[1]);
    if (type < 0 || type > 2)
    {   fprintf(stderr,"update:error¥n");
        exit(1);
    }
    key=atoi(argv[2]);
    if (key < 0 || key > 49)
    {   fprintf(stderr,"update:error¥n");
        exit(1);
    }
    if (type == 1)
        mf[key][1] *= 2;
    else if (type == 0)
        mf[key][1] /= 2;
    else
        printf("type number error¥n");
    printf("Update : Number %d was updated
        into %d¥n",key,mf[key][1]);

    fp=fopen("master","w");
    for (i=0;i<50;i++)
        fprintf(fp,"%d %d¥n",mf[i][0],
                mf[i][1]);
    fclose(fp);
}
```

図9 図8の通信ルーチンで呼ばれるC言語によるタスク

資源について

資源名	平均使用時間	平均使用率
ﾌﾗｸﾞ	4.000	0.760

資源の待ち行列について

平均待時間	平均待ち行列長
0.842	0.160

図10 性能評価結果

```
－－－－－－－－－－－－－－－－－－時刻 21 の動き
甲 1 甲ｴﾝﾄﾘｰ
乙 1 後処理 gg_
甲 2 ﾌｧｲﾙ更新
乙 2 前処理 gg_
乙 1 後処理 dd
乙 2 前処理 donh
乙 1 乙ｴﾝﾄﾘｰ d
乙 2 ﾌｧｲﾙ更新 i
－－－－－－－－－－－－－－－－－－時刻 21 の状態ｻﾏﾘ
処理要求(甲,1,甲ｴﾝﾄﾘｰ,17,[],[]).
処理要求(甲,2,ﾌｧｲﾙ更新,22,時間経過,[5,使用終了,ﾌﾗｸﾞ]).
処理要求(乙,1,乙ｴﾝﾄﾘｰ,21,[],[]).
処理要求(乙,2,ﾌｧｲﾙ更新,anyTime,待ち,[2,使用開始,ﾌﾗｸﾞ]).
資源状態(ﾌﾗｸﾞ,busy).
処理要求発生状態(甲,3,甲ｴﾝﾄﾘｰ,10,2,28).
処理要求発生状態(乙,3,乙ｴﾝﾄﾘｰ,10,2,29).
```

(A) 稼働状態のスナップショット

```
Update : Key 42 : Content 42 was updated into 84
Update : Key  7 : Content  3 was updated into  6
Update : Key 27 : Content 27 was updated into 54
Update : Key 14 : Content  7 was updated into 14
Update : Key 42 : Content 84 was updated into 42
Update : Key 15 : Content 15 was updated into 30
Update : Key 32 : Content 32 was updated into 64
Update : Key  7 : Content  6 was updated into 12
Update : Key 36 : Content 36 was updated into 72
Update : Key 33 : Content 32 was updated into 64
Update : Key 17 : Content 17 was updated into 34
Update : Key 28 : Content 28 was updated into 14
```

(B) ファイル更新の様子

図11 プロトタイプの振舞い