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††† 本位田真一††††

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．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ソフトウェアシステムである．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ためには，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い，このルートを除々に詳細化する中で，排他的なアクセスを必要とする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，個々の内部でのアルゴリズムや実行手順を設計する．この機能面での設計とともに性能面の設計の作業が並行処理ソフトウェアシステムに対して特に必要である．このためには，機能面での設計に並行し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りを正しく導入する．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．

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)）を総合して，筆者らは次の 4 点にまとめる．

（1） 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）は対象システムと異なっていてもよいが，最初から稼働可能であり，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．

（2） 環境導入性：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．

（3） 作成や変更の迅速性：対象システムのもつべき機能項目のうち，着目した項目の機能を迅速に装備可能であり，対象システムに対する仕様の変更，誤り，不明確点に迅速に対応できて変更可能である．

（4） 進化性：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．

図 1 に示すとおり，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ために，上記の四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ソフトウェアモジュールから成り，そ

† Prototyping Method and Tool for Designing Concurrent Processing Software Systems by YASUHISA TAMURA, KIYOSHI ITOH (Faculty of Science and Technology, Sophia University) and SHINICHI HONIDEN (Toshiba Corporation).
†† 上智大学大学院理工学研究科機械工学専攻修士課程
††† 上智大学理工学部一般科学研究室情報科学部門
†††† (株)東芝システム・ソフトウェア技術研究所

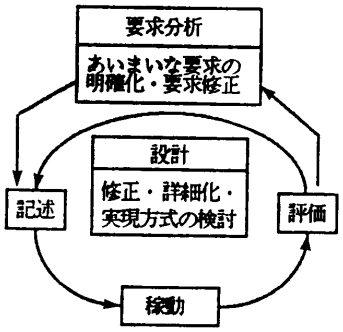

図 1 ソフトウェアプロトタイピングサイクル
Fig. 1 Concept of software prototyping cycle.

れらが並行して多数のトランザクションを処理するソフトウェアシステムである．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ともに，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．これらの二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．

以下の章では，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．

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合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さらに分離・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とも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—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—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—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，資源の稼動率など—が設計上実現可能か否かを調べることである．

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

ステップワイズ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，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，明確に分解して浮かび上がらせていく次のプロセスである．

＜フロー指向パラダイムによるプロトタイピング手順＞

（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，一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すべ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りあるいは空間的な使用量を再検討する．

以上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，設計が進展すると，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ための，段階的な置換えと詳細化の過程である次のプロセスを行う．

＜タスク指向混在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りを再検討する．

ステップワイズプロトタイピングによる設計過程を**図2**に示す．この図は，性能要求項目の制約—性能面での要求項目を満たしているか否か—の下に，機能要求項目について詳細化しながら設計を進めていく—処理要求や機能モジュールを数え上げ，それらの動的な関係を明らかにする—プロセスを，概念的に示している．この図左方で，丸く書かれたものや破線で書かれたものは，フロー指向パラダイムで書かれた機能モジュールおよび互いの関係で，初めはあまり明確にはなっていないことを示す．設計の進展につれて，機能モジュールとして認識され，機能モジュール間の関係，

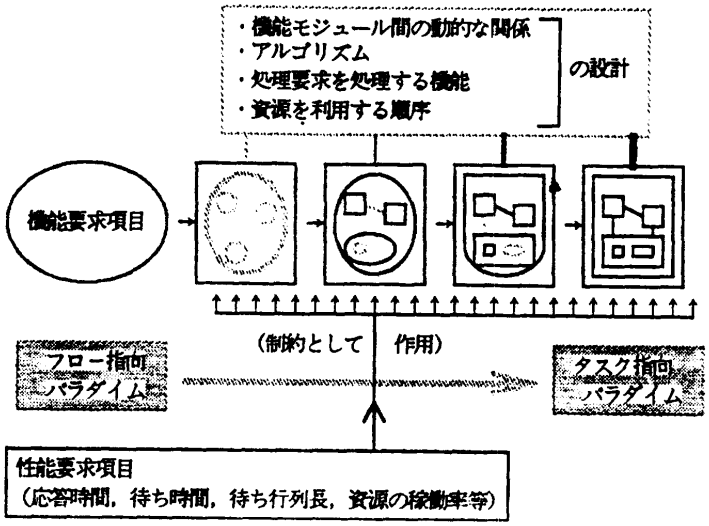

図 2 並行処理ソフトウェアシステムの設計
Fig. 2 Design concept of concurrent software system.

アルゴリズム，資源の利用順序などが明らかになるが，これをこの図右方で四角く書かれたものや実線で書かれたもので示す．

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．

プロトタイプを稼働させながら，その機能のテストを行う．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や，機能モジュールや資源の利用の正しさの確認により行われる．また，並行処理ソフトウェアシステム内でのデッドロックの検出も含まれる．また，プロトタイプの性能評価は，システム中の処理要求やタスク数の測定，処理要求の待ち行列長や利用される資源の量（平均稼働率など）・時間（平均使用時間など）の測定などにより行われる．

以上の機能のテストや性能の評価により，良好な結果が得られた場合には，ステップワイズプロトタイピングの手順に従い，さらに詳細化を進めてよい．不満足な結果の場合にもステップワイズプロトタイピングの手順に従い，必要な修正を施す．

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，一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，一つの事実が 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．一つの処理要求が複数の属性をもつことは，一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，一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，すべて Prolog 組込み述語を使って 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，処理要求のフローの正しさ，待ち行列の形成，デッドロックの発生などの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，処理要求の個数や待ち行列長，資源や機能モジュールの平均稼働率などの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 言語構成要素とその構文法

P-Flots 言語の構文を図 3 に示す．

3章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，機能モジュールごとに，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設計の各段階で，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．

```
<設計記述>
      ← {<処理要求発生文>.}₁ {<機能モジュール記述>}₁ .
        {<Prolog規則形式述語>}₀ . {資源状態(<資源名>, <状態>).}₀ .
        時刻(<int>) . 終了時刻(<int>) .
<処理要求発生文> ← 処理要求発生(<処理要求名>, <int>, <int>,
                                <機能モジュール名>)
<機能モジュール記述>
   ← 機能要素 (<機能モジュール名>, <処理要求名リスト>, <流れ要素>)
    {. 機能要素 (<機能モジュール名>, <処理要求名リスト>, <流れ要素>)}₀
<Prolog規則形式述語> ← <規則名>
          | <規則名> :- <組込み述語> {, <組込み述語>}₀
<流れ要素> ← <単純流れ要素> | [<複合流れ要素>]
<複合流れ要素> ← <if流れ要素> | <select流れ要素> | <while流れ要素>
                | <repeat流れ要素>
<if流れ要素> ← [if, <条件>] {, <単純流れ要素>}₁
                {, [else] {, <単純流れ要素>}₁}₀
<while流れ要素> ← [while, <条件>] {, <単純流れ要素>}₁
<repeat流れ要素> ← [repeat] {, <単純流れ要素>}₁ , [until, <条件>]
<select流れ要素> ← [select] {, [<条件>, <擬似タスク名>]}₁
<単純流れ要素> ← [時間経過, <int>] | [時間経過, [<int>, <int>]]
          | [使用開始, <資源名>] | [使用終了, <資源名>]
          | [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>]]
          | [ﾒｯｾｰｼﾞ受信, [<ﾒｲﾙﾎﾞｯｸｽ名>, <ﾒｯｾｰｼﾞ内容>]]
          | [実行, <機能モジュール名>] | [実行, <規則名>]
<状態> ← free | busy
<組込み述語> ← <P-Flots組込み述語> | <Prolog組込み述語>
<処理要求名リスト> ← [<処理要求名> {, <処理要求名>}₀] | [_]

<資源名>，<機能モジュール名>，<処理要求名>，<条件>，<ﾒｲﾙﾎﾞｯｸｽ名>，
<ﾒｯｾｰｼﾞ内容>，<擬似タスク名>，<P-Flots組込み述語>，<Prolog組込み述語>，
<規則名>は文字列．　<int>は整数．

<と>で囲まれたものは非終端記号，<と>で囲まれていないものは終端記号，
{α}₁はαの1回以上の繰り返し，{α}₀はαの0回以上の繰り返し，α | βはαまたはβの選択を示す．←は左辺の非終端記号が右辺で示されたものに置き換え可能であることを示す．
```

図 3 P-Flots 言語の構文
Fig. 3 Constructs of P-Flots language.

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，処理要求の種類ごと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「ファイル使用」と「後処理」を使用する，という形で記述する（図 4 (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

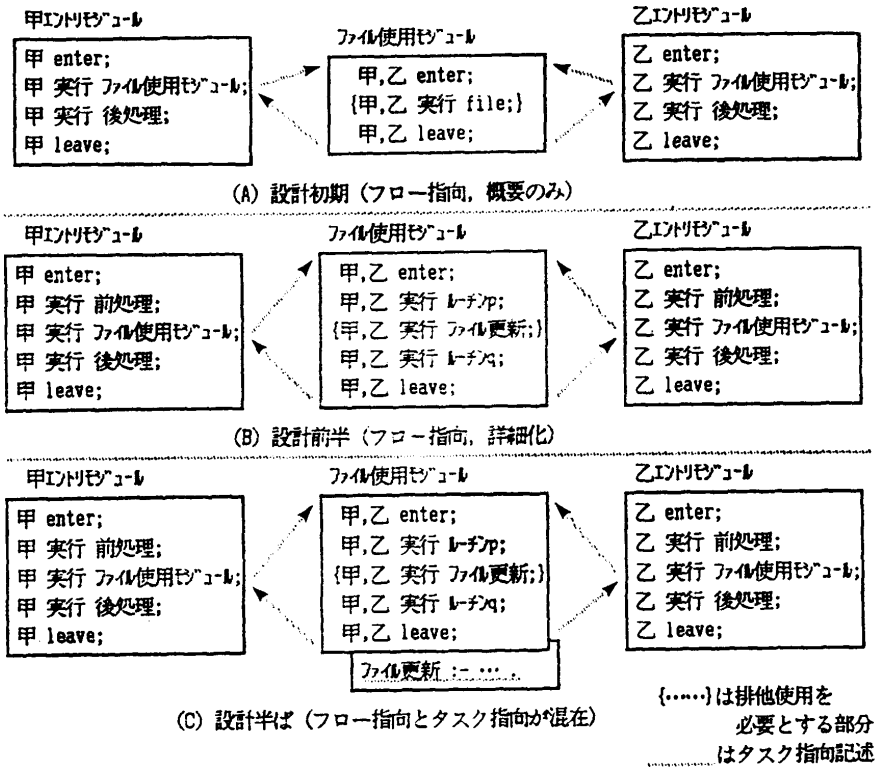

図 4 システムの設計例
Fig. 4 Design example.

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，の三つが明らかになったとする（図4（B））．この段階における設計者の P-Flots 記述を**図5**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．

```
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,ﾌｧｲﾙ使用]).
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,ﾌｧｲﾙ使用]).
機能要素(乙ｴﾝﾄﾘｰ,[乙],[実行,後処理]).

機能要素(ﾌｧｲﾙ使用,[_],[実行,ﾙｰﾁﾝp]).
機能要素(ﾌｧｲﾙ使用,[_],[使用開始,ﾌﾗｸﾞ]).
機能要素(ﾌｧｲﾙ使用,[_],[実行,ﾌｧｲﾙ更新]).
機能要素(ﾌｧｲﾙ使用,[_],[時間経過,4]).
機能要素(ﾌｧｲﾙ使用,[_],[使用終了,ﾌﾗｸﾞ]).
機能要素(ﾌｧｲﾙ使用,[_],[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 5 図4(B)の P-Flots 記述
Fig. 5 P-Flots description of Fig. 4(B).

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4（C）と図5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び出す記述が**図6**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

```
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 6 呼び出される Prolog 規則およびタスク呼出し
Fig. 6 Called prolog rule and calling task.

「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．このタスクはC言語で記述されている．

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 7 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 8 に示す．

```
資源について
資源名    平均使用時間    平均使用率
フラグ      4.000          0.760
資源の待ち行列について
平均待ち時間    平均待ち行列長
   0.842          0.160
```

図 7 性能評価結果
Fig. 7 Performance result.

```
----------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
(A) Snapshot in Execution

```
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
(B) History of File Update

図 8 プロトタイプの振舞い
Fig. 8 Behavior of the prototype.

図 9 にデッドロックが起きる P-Flots 記述例を示

```
処理要求発生(甲,10,3,甲ﾒｲﾝ).
機能要素(甲ﾒｲﾝ,[甲],[使用開始,disk1]).
機能要素(甲ﾒｲﾝ,[甲],[時間経過,[3,1]]).
機能要素(甲ﾒｲﾝ,[甲],[使用開始,disk2]).
機能要素(甲ﾒｲﾝ,[甲],[時間経過,2]).
機能要素(甲ﾒｲﾝ,[甲],[使用終了,disk2]).
機能要素(甲ﾒｲﾝ,[甲],[時間経過,1]).
機能要素(甲ﾒｲﾝ,[甲],[使用終了,disk1]).

処理要求発生(乙,9,4,乙ﾒｲﾝ).
機能要素(乙ﾒｲﾝ,[乙],[使用開始,disk2]).
機能要素(乙ﾒｲﾝ,[乙],[時間経過,4]).
機能要素(乙ﾒｲﾝ,[乙],[使用開始,disk1]).
機能要素(乙ﾒｲﾝ,[乙],[時間経過,3]).
機能要素(乙ﾒｲﾝ,[乙],[使用終了,disk1]).
機能要素(乙ﾒｲﾝ,[乙],[使用終了,disk2]).

資源状態(disk1,free).
資源状態(disk2,free).
時刻(0).
終了時刻(200).
```

図 9 デッドロックを引き起こす P-Flots 記述
Fig. 9 P-Flots description involving deadlock.

```
----------時刻 18 の動き
乙 1 乙ﾒｲﾝ i
甲 1 甲ﾒｲﾝ i
乙 2 乙ﾒｲﾝ i
甲 2 甲ﾒｲﾝ ch
----------時刻 18 の状態サマリ
処理要求(乙,1,乙ﾒｲﾝ,anyTime,待ち,[3,使用開始,disk1]).
処理要求(甲,1,甲ﾒｲﾝ,anyTime,待ち,[3,使用開始,disk2]).
処理要求(乙,2,乙ﾒｲﾝ,anyTime,待ち,[1,使用開始,disk2]).
処理要求(甲,2,甲ﾒｲﾝ,anyTime,待ち,[1,使用開始,disk1]).
資源状態(disk2,busy).
資源状態(disk1,busy).
処理要求発生状態(乙,3,乙ﾒｲﾝ,9,4,20).
処理要求発生状態(甲,3,甲ﾒｲﾝ,10,3,30).
```

図 10 図 9 の記述を稼動させたときのデッドロックに陥った状態のスナップショット
Fig. 10 Snapshot at deadlock state by Fig. 9.

す．図10はそのデッドロックが起きた後のスナップショットを示す．この中で，甲の1番目の処理要求がdisk 1をつかんだ後，disk 2の利用待ちになっており，一方，乙の1番目の処理要求がdisk 2をつかんだ後，disk 1の利用待ちになっている．この二つの処理要求がデッドロック状態になっている．その他の後続の処理要求は何もつかめないまま，待ち状態になっている．このように設計者はプロトタイプの稼働状態を視認できる．

6. むすび

P-Flotsにより開発されたプロトタイプが1章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，作成や変更の迅速性，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び出すことによって，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．

述語に並行性をもたせ，それらの述語間のスケジューリング機能，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．処理要求の時間経過やスケジューリングなどをT-Prologの処理系に任せれば，P-Flotsのインプリメンテーションがより簡単になると考えられる．しかし，筆者らは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ピングを可能とすることが重要であると考え，P-Flotsの現バージョンのインプリメンテーションで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IBM 5540およびIBM 5560 (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．IBM 5540はフロー指向の設計作業に，IBM 5560はタスク指向の設計作業に使われている．

IBM 5540の主記憶は640 KBで，DOS，Prologインタプリタ，P-Flotsのインタプリタ，P-Flots言語によるプロトタイプを配置している．このため稼働できるプロトタイプの規模はP-Flots言語で約200行である．またこの配置のためプロトタイプの稼動の効率や応答性は一般的に必ずしも良好ではない．ただし，5章で述べた程度の規模のプロトタイプの稼動では，利用者はほとんど待たされずP-Flotsは十分な応答性をもつ．P-Flots言語で記述したプロトタイプを，1章で述べたGPSSをベースとして大型計算機上で稼動できるG-Flots向きのプロトタイプに変換するトランスレータを別途開発中である．これによりプロトタイプの規模に応じてP-FlotsとG-Flotsを切り換えて，記憶容量や稼働の効率に対する制約に対処したいと考える．

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．

謝辞 本研究は，一部文部省科学研究費昭和60年度奨励研究（A）(60780053) による．

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 (Mar. 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一手法，情報処理学会論文誌，Vol. 26, No. 5, pp. 946-953 (Sep. 1985).
4) Intermedia Access Corporation: Prolog-J (Mar. 1985).
5) Itoh, K. et al.: Software Design Process: Chrysalis Stage under the Control of Designers, *J. Inf. Process*, Vol. 7, No. 1, pp. 5-14 (Mar. 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 (Aug. 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 Engineering Note*, Vol. 7, No. 5 (Dec. 1982).
11) Tamura, Y. and Itoh, K.: 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*, Vol. 9, No. 4, pp. 12-19 (1984).

(昭和61年8月11日受付)
(昭和62年6月11日採録)

田村 恭久 (正会員)

昭和36年生. 昭和60年上智大学理工学部機械工学科卒業. 昭和62年同大学院理工学研究科機械工学専攻修士課程修了. 同年, (株)日立製作所に勤務. 同システム開発研究所所属. 在学中は, プロトタイピング手法とツールおよびシミュレーション手法の研究に従事. ソフトウェア工学全般と AI に興味をもつ.

伊藤 潔 (正会員)

昭和26年生. 昭和49年京都大学工学部情報工学科卒業. 昭和51年同大学院工学研究科情報工学専攻修士課程修了. 昭和54年同博士課程修了. 工学博士. 昭和54年より上智大学理工学部に勤務. 助手, 講師を経て, 昭和60年より助教授. 現在, 同理工学部一般科学研究室情報科学部門所属. 主として, ソフトウェア工学, シミュレーション手法, 形状処理工学の研究に従事. 計算機援用の様々なシステム・手法に興味をもつ. 電子情報通信学会, IEEE 等会員. ISO/TC 184/SC 2 WG 3 (産業用ロボットの安全性) 委員.

本位田真一 (正会員)

昭和28年生. 昭和51年早稲田大学理工学部電気工学科卒業. 昭和53年同大学院理工学研究科電気工学専攻修士課程修了. 工学博士. 同年東京芝浦電気(株)入社. 現在, (株)東芝システム・ソフトウェア技術研究所研究主務. 主として, ソフトウェア工学, 人工知能の研究に従事. ソフトウェアの基礎理論に興味をもつ. 昭和61年度情報処理学会論文賞受賞. 電気学会, AAAI 各会員.