ソフトウェア工学 98－8
（1994．5．25）

プロセス・プロダクトの関係に基づく プロセス記述・実行支援システムの設計

上田賀一，志村秀人，安間その美

茨城大学 工学部 情報工学科

コンピュータ上での作業負荷の軽減，効率化を図るため，作業指針を示すプロセスを形式的に記述し，これに従って作業を進める試みがなされている．本研究では，現実的なプロセス記述／実行を実現するために，開発および利用の異なるフェーズにおける観点とプロセスとプロダクトの両側面の観点からプロセスの特性を捉え，それに基づいた支援方法を考える．また，プロセスを簡潔に記述し，容易に理解できるようにするために，ダイアグラム形式のプロセス記述法を用意し，メタルールに基づくプロセスの簡略記述を提案する．さらに，プロセス記述／実行支援システムを設計し，その試作について考える．

Process description and execution supporting system based on relations among processes and products

Yoshikazu Ueda, Hideto Shimura, Sonomi Anma

Ibaraki University

To decrease the load and increase the efficiency in working on the computer system, there are many researches about the process description which guides the worker to the reasonable jobs. In our study, we investigate the characteristics of the process at the viewpoints of the development phase and the use phase. And the viewpoints of the processes and the products, too. Then we study the supporting method based on the characteristics. Further, we show the process description language which is diagram style, and propose the simple description method based on the meta-rules. Additionally, we design the process description / execution supporting system, and consider its implementation.

1．はじめに

通常，コンピュータ上で何らかの作業をする際には，作業者が自分自身で作業内容を考えなければならない．しかし，コンピュータの使用経験の少ない作業者にとってこれは大きな負担となる．実際，UNIX 環境や DOS 環境におけるソフトウェア開発や情報処理では，種々のツールを用い，いくつものファイル変換を経て，いくつものファイルの組み合わせて進めていく形態を採る．この際に扱うツールやファイルの間には様々な論理関係があり，開発者や利用者に対する制約となって表出する．そこで，作業者の負担を軽減し，作業の効率をあげるために，行うべき作業に対する指針を作業者に提供することが求められる．

現在，これらの作業指針を示すプロセスを形式的に記述し，この記述に従って作業を進める試みがなされている．これによって，プロセスの管理や保守や再利用がしやすくなる．開発者にとっては，プロセスをソフトウェア開発工程とみなすと，プロセス記述によって，開発に携わる人が自分自身のプロセスを曖昧さなく理解できるだけでなく，効率的なソフトウェア開発支援を行うことができる．また，利用者にとっては，プロセスは処理過程であり，プロセス記述が提供されることで，処理手順に悩まされることなく，処理を進めることができる．

これまでに提案されているプロセス記述の方法は，テキスト記述のもの[1][2]が多かったが，最近では，図式表現を用いてプロセスを定義するものもある[3][4]．記述したものを実行できる，あるいは解析できるという点で，状態遷移図，ペトリネットなどのネット図でプロセスをモデル化し，記述する研究も盛んである[5]．

また，コンピュータ支援の観点からでは，プロセスを中心に考えるものばかりでなく，プロダクトを中心に考える手法もある．

本研究では，次の3点を目的としたプロセス記述／実行支援システムの構築を目指す．

(1) 開発および利用の異なるフェーズでのプロセスの記述や実行を支援すること．
(2) プロセスを図式的に記述し実行できる方法と環境を提供すること．
(3) 記述したプロセスは理解容易であり，簡便に管理，利用できること．

本稿では，始めに，開発および利用という2つの異なるフェーズにおけるプロセスの記述・実行に関する特性を捉え，それに基づいた支援方法を考える．次に，プロセスとプロダクトの両側面から現実的なプロセス記述／実行を実現するための可能な領分を見極め，より容易および簡潔に扱うための指針を検討する．また，視覚的効果の高いインタフェースに基づくツールとするため，プロセス記述ではダイアグラム形式の記述法を用意する．さらに，プロセスを簡潔に記述し，容易に理解できるようにするために，メタルールに基づくプロセスの簡略記述を提案する．最後に，プロセス記述／実行支援システムを設計し，その試作について考える．

2．プロセスの開発と利用

プロセスとプロダクトが密接に関連していることは明らかなことである．ソフトウェアシステムに関しては両側面から理解できることが望まれるが，実際にはそういかないものである．プログラマや設計者といった開発者は処理手順や作業手順といったプロセスの観点で計算機環境上での活動や仕事を捉えることに慣れているが，そうでない利用者はアプリケーションファイルやデータファイルといったプロダクトの観点で捉えることで活動や仕事を理解していることが多いと考えられる．

そこで，本研究では，プロセスを開発および利用という2つの異なるフェーズに分けて考えることにする．このフェーズ分けされたプロセスはオブジェクト指向のクラスとインスタンスに対応付けることができる．開発者側では，利用者側がどのようなアプリケーションを有し，どれを利用しているのか想定できないので，抽象的なレベルでプロセスを記述せざるを得ない．このプロセスを提供された利用者側では，具体的なアプリケーションを割り当てること，即ち，インスタンス化

することによりプロセスを実行できることになる．例えば，開発者側がプロセス中の編集作業に対しどのようなエディタが利用者側に準備されているか分からない場合には，単にテキストエディタというアプリケーションクラスとしてプロセスを記述することになる．利用者側は，テキストエディタとして vi や emacs をインスタンスアプリケーションとして用いることにより，このプロセスの実行を進めることができるようになる．こうして，各プロセスのプロダクトをオブジェクトと捉える一貫した考え方の採用で，異なるフェーズのプロセスを容易に理解できる形で結び付けることができる．

ここで，改めてクラスプロセスとインスタンスプロセスを定義しておく．

一連の作業系列中に一つの実体の定まらないクラスアプリケーションが一つ以上存在する系列をクラスプロセスという．また，一連の作業系列中の全てのアプリケーションに対し，それぞれが一つの実体に定まっている系列をインスタンスプロセスと呼ぶ．開発者側はクラスプロセスだけに触れ，利用者側はクラスプロセスとインスタンスプロセスのいずれにも触れる．然るに，開発者側の作業はプロセスの記述だけである．利用者側はプロセスのインスタンス化という記述作業が必要となり，その後，プロセスの実行となる．この記述作業はプロセスそのものの記述というより，クラスアプリケーションに対し，インスタンスアプリケーションを一つもしくは複数割り当てる作業となり，プロダクトの観点で容易にこの作業を行うことができる．この際，インスタンスアプリケーションを割り当てる個数によって後の実行方式に差異が生じる．それが，自動実行と対話実行の２方式である．一意に定まったアプリケーションは自動実行され，複数の実体を持つアプリケーションは選択を行うことで実行される対話実行となる．

以上，開発と利用の側面で分けることにより，それぞれのフェーズの特性に応じた作業内容でプロセスの記述や実行ができるようになる．また，このことは開発環境と利用環境の違いを吸収する役割を果たすこともできる．

3．プロセスとプロダクト

一般にプロセスはアプリケーションとデータの並びからなる一連の系列と考えることができる．部分的には「入力 - 処理 - 出力」といった入出力データとアプリケーションの列と捉えられる．この際，アプリケーションにとって入力データは必須のものや必ずしも要らないものがあり，出力データは必ず生成されるものや場合によって生成されるものがある．当然，データにとってその存在は入力関係にあるアプリケーションの起動要因となり得るし，入出力関係にないアプリケーションを起動したいこともある．これらのことは，プロセス記述言語に必須／選択の区別，データおよび制御の流れを取り入れなければならないことを示唆する．

前述したように，利用者にとって計算機環境上での作業はプロダクトの観点で捉えられた方が理解しやすいと言える．そこで，利用者が理解できる範囲でプロダクトを分類し，その情報をできるだけプロセス記述に役立てることを考える．まず，プロダクトをアプリケーションとデータに分ける．入出力データとアプリケーションの関係は入力と出力のデータが同一のプロダクトとなる編集処理，入力と出力のデータが異なるプロダクトとなる変換処理，確認などのために入力データをディスプレイやプリンタに出力する表示処理に大別できる．これらはアプリケーションの分類として，エディタ型，フィルタ型，ビューア型と呼ぶことにする．一方，データはプロダクト管理の観点から永続データと一時データに分類する．

このアプリケーションの分類は，データの流れや制御の流れを自然に（常識的に）制限付ける役割を果たす．この制限付けによりプロセス記述を簡潔なものとすることができ，記述および理解の容易性を高めることに結び付く．本研究では，これをアプリケーションに基づくプロセス記述のためのメタルールとして準備することにした．メタルールの詳細については後述する．また，このデータの分類は，データの必須／選択の区別を明確にするものではないが，プロダクトの重要度に対する主要因として働き，これだけでも記述の手間がかなり省けると考える．しかし，明確に必須／

選択を示す場合のために，記述法においてこれをサポートしておく．

次に，プロセスの実行について考える．この際，プロセスの実行・管理支援システムあるいはツールは，種々のアプリケーションを起動し，それらとの情報伝達を必要とする．しかし，現状においてアプリケーションはプロセスの一部として実行され管理されるものとして存在しているわけではない．だからといって，プロセスの実行・管理の目的の下に現存の多数のアプリケーションを作り直すことは大変な作業であり，現実的な方法ではない．そこで，完全な情報伝達ではなくても，現存のアプリケーションから得られる現実的な情報に基づくプロセスの実行・管理を考えることにする．ところで，現状のアプリケーションはツールなのか環境なのか区別できないほどその機能は向上している．しかも，マルチウィンドウシステムの普及などもあり，エディタやビューアを終了することなく，次の作業に進むことができ，また，これらを常駐しておきたいのが実情である．この状況において得られる情報を挙げると，アプリケーションの終了，エディタ型アプリケーションによるデータファイルの生成および更新，フィルタ型アプリケーションによるデータファイルの生成，フィルタ型アプリケーションによる処理の失敗がある．これらのイベントを契機としてプロセスを進行させることができる．それ以外のプロセスの進行はユーザの意志に基づくものとし，エディタ型およびビューア型アプリケーションの利用時における進行可能な作業の選択はメニュー操作による対話方式でサポートする．

4．プロセス記述

4.1　プロセス記述法

本研究では，できるだけ簡潔で理解容易な記述法とするために，プロセス記述にはダイアグラム形式を用いることにする．特にデータフローモデルに基づいたプロセス記述をベースとし，オブジェクト指向とプロダクトの観点から捉えやすいものを目指した．その結果，図1に示す9種類のノードと2種類のアークからなるダイアグラム表記法とした．各ノードを以下に説明する．

○ノード

・永続データ

プロセス実行後も永続的に保存するデータファイルの抽象概念を表すノードであり，入出力のアークはそれぞれ一つずつでなければならない．また，実行時には単に通過点となり，このノードに対して生成されたインスタンス（データファイル）は実行後もシステム上に保存される．

・一時データ

プロセス実行中だけ存在し，実行後は保存しておく必要のないデータファイルの抽象概念を表すノードであり，入出力のアークはそれぞれ一つずつでなければならない．また，実行時には単に通過点となり，このノードに対して生成されたインスタンス（データファイル）は実行後に削除される．

・クラスアプリケーション

アプリケーションの抽象概念を表すノードであり，入出力のアークは基本的に一つずつでなければならない．また，実行時には，入力が来たらインスタンス（実行ファイル）を呼び出し，処理を行う．そして，処理が終了したら次のノードへ出力する．このノードはさらにエディタ型，フィルタ型，ビューア型の3種類のものがある．

・インスタンスアプリケーション

アプリケーションの実体を表すノードであり，入出力のアークは基本的に一つずつでなければならない．また，実行時には，入力が来たら処理をし，処理が終了したら次のノードへ出力する．このノードもさらに3種類のものがある．

・OR 分割

あるデータまたはアプリケーションからの出力が複数あり，その中のどれかが有効となる場合にそのノードの後にこのノードを置いて使用する．この場合，このノードの入力のアークは一つでなければならないが，出力のアークはいくつでもよい．また，実行時には，入力に対し受け取り可能な出力先のすべてに出力する．

・OR 結合

あるデータまたはアプリケーションへの入力が複数あり，その中のどれかが有効となる場合にそのノードの前にこのノードを置いて使用する．この場合，このノードの出力のアークは一つでなければならないが，入力のアークはいくつでもよい．また，実行時には，入力のどれか一つに対し出力する．

・AND 分割

あるデータおよびアプリケーションからの出力が複数あり，その中のすべてが有効となる場合にそのノードの後にこのノードを置いて使用する．この場合，このノードの入力のアークは一つでなければならないが，出力のアークはいくつでもよい．また，実行時には，入力に対し出力先のすべてに出力する（このノードを利用すればプロセスの並列実行が記述できる）．このノードは，アプリケーションとデータ間を１対多あるいは多対１で直接結び付けることにより省略できる．

・AND 結合

あるデータおよびアプリケーションへの入力が複数あり，その中のすべてが有効となる場合にそのノードの前にこのノードを置いて使用する．この場合，このノードの出力のアークは一つでなければならないが，入力のアークはいくつでもよい．また，実行時には，すべての入力が揃うことで出力する（このノードを利用すれば同期を取ることができる）．このノードも，アプリケーションとデータ間を１対多あるいは多対１で直接結び付けることにより省略できる．

・条件分岐

条件によってアーク先が異なる場合に使用し，入力のアークは一つでなければならないが，出力のアークはいくつでもよい．ただし，入力と出力のデータは同じものでなければならない．また，実行時には，アーク先の選択をユーザに促す．ただし，実行時において一意に決定できる場合は自動的に分岐する．したがって，このノードはスイッチのような役割を果たす．

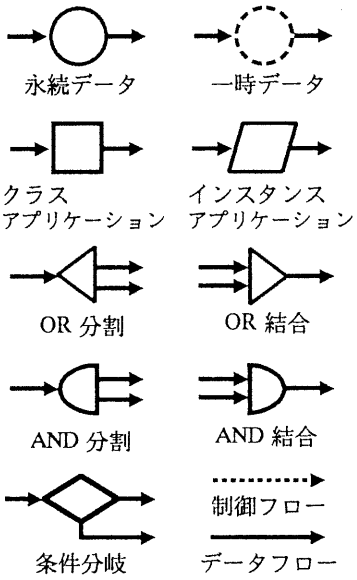


図１．プロセス記述法

○アーク

・データフロー

アプリケーションとデータのノードを結び付ける有向線分であり，アプリケーションに対する入出力を表す．また，他のノードに対しても直前のデータノードからのデータの流入を意味し，直後のノードへの同一データの流出を意味する．

・制御フロー

アプリケーションのノード間を結び付ける有向線分であり，有向線分元のアプリケーションにおける何らかのイベントに対し，有向線分先のアプリケーションに制御を移すことを意味する．

4.2 メタルール

さらに本研究では，プロセス記述の簡便化を図り，理解容易性を高めるために，常識的に判断できる流れは記述を省略できるようにすることを考えた．特に，利用者にとって理解しにくい制御の流れは極力省略できることが望ましいと考えた．そこで，メタルールに基づくプロセスの簡略記述を提案する．メタルールは，３つの型に分類したアプリケーションとデータとの間での可能な制御

とデータの流れを制限するものであり，利用者にとっては常識的な作業手順をルール化したものである．図2にプロダクト間の可能な制御とデータの流れを示す．本研究が取り上げた常識的な作業手順によるルールには以下のものがある．

(1) フィルタ型アプリケーションにおける処理の失敗は，直前のエディタ型アプリケーションのやり直しとなる．

(2) ビューア型アプリケーションの中断や終了は，直前のエディタ型アプリケーションへの戻りかプロセスの終了となる．

(3) データの生成や更新は次のアプリケーション起動の必要条件となる．

(4) フィルタ型アプリケーションだけからなる一連の部分系列は一つのブロックとして扱うことができ，途中のデータは全て一時的なものであってもよい．

(1) については，ソフトウェア開発においては必ず取られる作業手順であるので，わざわざプロセスに制御の流れとして記述する必要のないものである．また，大半の制御の流れはこれに属するものなので，これをメタルールにすることで多くの制御の流れの記述を省くことができる．

(2) については，通常，ビューア型アプリケーションはプロセスの終端にあるので，この次の作業は終了するか前戻りのいずれかの選択となってしまう．そこで，メニュー操作による次の作業選択が必要となる．

(3) については，十分条件ではなく必要条件であることに注意すべきである．エディタなどを終了せずにデータの書き出しを行った場合，それが次の作業への進行を意図したものかどうかは分からないからである．しかし，利用環境の能力が余裕をもって高い場合には，意図に関係なく背景で処理を進行させることができる．よってこの作業進行には自動か対話の大局的な前提の設定が必要である．

(4) については，プロセスの記述上，何ら利点になるものではないが，実行上においてはひとまとめのアプリケーションとして扱うことができ，本システムの実現において有効な処理単位となる．

上記のメタルールに基づき，簡略記述されたプロセス図から実プロセスモデルへの変換機能をプロセス記述システムに採り入れることで，先の目的を果たすことにした（図3）．

本研究が扱うプロセスの例を示す．図4はCプログラムの開発作業であり，図5は LaTeX によるレポート作成作業である．いずれも，プロセス記述は上段の簡潔なものでよく，変換機能により下段の実プロセスに変換し，これに実体のアプリケーションを割り当てることで実行される．

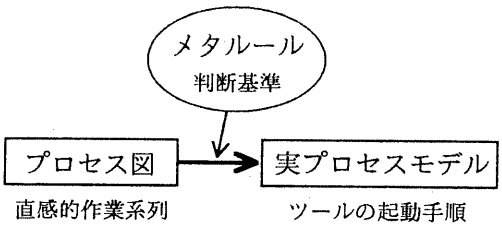


図3．簡略記述からの変換

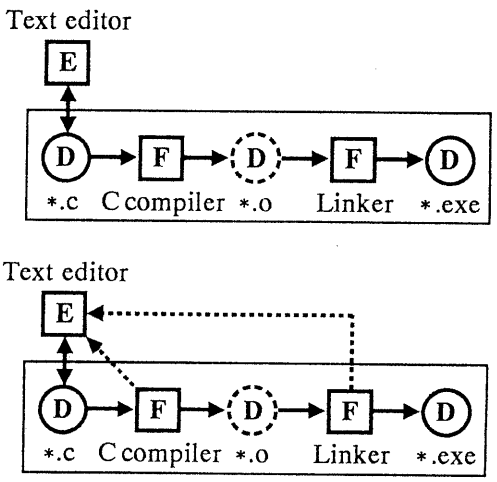


図4．C プログラム開発のプロセス記述例

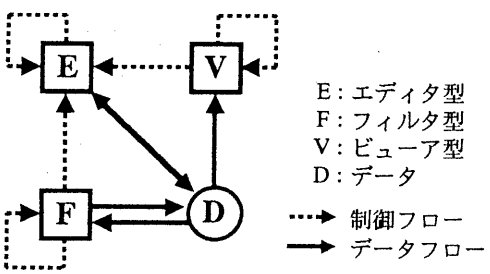


図2．メタルール（プロダクト間の関係）

5．プロセス記述／実行支援システムの設計と試作

5.1　システムの構成

本研究のプロセス記述／実行支援システムの構成概要を図６に示す．本システムは６つの支援ツールから構成され，５つのファイル(群)を取り扱う．以下に支援ツールとファイル(群)について説明する．

○支援ツール

・コンポーネントエディタ (CE)
　コンポーネントに関する情報を記述するためのもので、コンポーネント情報ファイル (CF) を出力する。

・プロセス図エディタ (PE)
　プロセスを図式的に記述・編集するためのものである。また、図式表現をテキスト表現に変換し、プロセス図情報ファイル (PF) を出力する。

・トランスレータ (TR)
　コンポーネント情報ファイル (CF) とプロセス図情報ファイル (PF) を統合し、その情報を実行情報ファイル (EF) として出力する。統合の際にはメタルールを適応し，簡略記述モデルを実プロセスモデルに変換する。

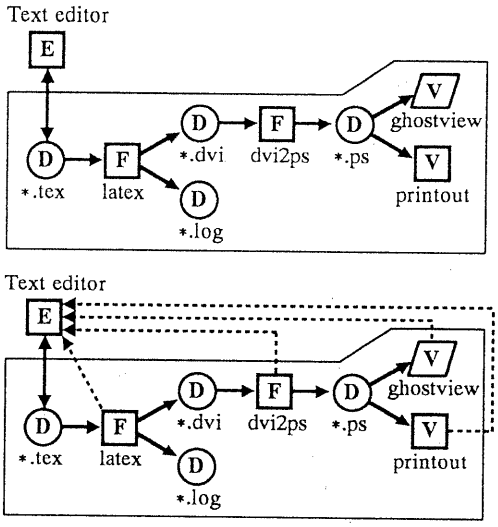
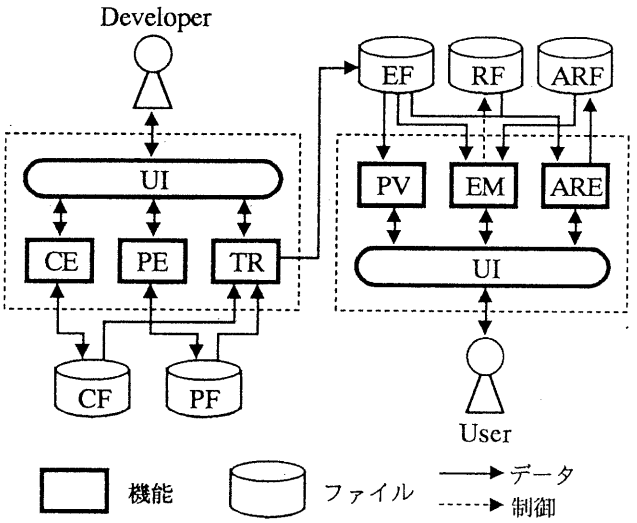


図５．レポート作成のプロセス記述例

・プロセスビューア (PV)
　プロセスの実行状況を図式的に表示するためのものである。

・実行マネージャ (EM)
　プロセスを起動し、実行を管理するためのものである。また、実行の際に扱うプロダクトについても管理する。



UI　: User Interface
CE　: Component Editor
PE　: Process diagram Editor
TR　: TRanslator
PV　: Process Viewer
EM　: Execution Manager
ARE : Application Registration Editor
CF　: Component information File
PF　: Process diagram information File
EF　: Execution information File
ARF : Application Registration File
RF　: Reguler File

図６．プロセス記述／実行支援システムの概観構成図

・アプリケーション登録エディタ (ARE)
プロセスをユーザの環境に適用するために、特定のクラスアプリケーションに対してインスタンスアプリケーションを登録し、その情報をアプリケーション登録ファイル (ARF) に出力する。

○ファイル

・コンポーネント情報ファイル (CF)
プロセスを構成するコンポーネント（アプリケーションやデータ）を管理するためのファイルのことである。このファイルにはコンポーネントに関するファイル名，ファイル種別などの情報が記述されている。

・プロセス図情報ファイル (PF)
プロセス図を管理するためのファイルのことである。このファイルにはノードの連結や位置などの情報が記述されている。

・実行情報ファイル (EF)
コンポーネント情報ファイル (CF) とプロセス図情報ファイル (PF) を統合したもので、プロセスを実行する際に必要となる情報が記述されているファイルのことである。

・アプリケーション登録ファイル (ARF)
プロセスをユーザの環境に適用するために、特定のクラスアプリケーションに対するインスタンスアプリケーションを登録するファイルのことである。

・通常ファイル (RF)
通常使用しているファイルのことである。通常のデータファイルやアプリケーションファイルがこれに含まれる。

5.2 システムの試作

実現に関しては，開発環境と利用環境が共にUNIX/X-Window 上であることを想定しており，Sparc Station, Sun OS 4 において本システムの試作を行っている．開発言語は主にC言語を用いている．システムを構成するツール群はツール間およびツール内のスレッド間の情報のやり取りや同期を取る必要があるので，その機能を実現するためプロセス間通信を用いる．また，より使いやすいものとするためには，プロセス実行に柔軟かつ動的に対応できるメニューやダイアログボックスによるユーザインタフェースが必要とされるので，この部分の構築には Tcl / Tk のライブラリを利用している．

6．まとめ

本研究では，プロセス記述・実行にあたり，開発および利用の異なるフェーズを設定し，各特性に応じた支援方法を考えた．また，プロセスとプロダクトの両側面から現実的なプロセス記述／実行を実現するための指針を検討した．さらに，プロセスを簡潔に記述し，容易に理解できるようにするために，ダイアグラム形式による記述法とメタルールに基づくプロセスの簡略記述を提案した．最後に，プロセス記述／実行支援システムを設計し，その試作について紹介した．今後，実用性の高いものに拡張していくと共に，開発と利用のフェーズ分けを活かし，開発環境と利用環境が大きく異なる場合への対応を考えていきたい．

参考文献

(1) 荻原，井上，鳥居：「ソフトウェア開発を支援するツール起動自動制御システム」，信学論 D-I，Vol.J72-D-I，No.10，pp.742-749（1989）

(2) 松永，荻原，井上，鳥居：「全体プロセス記述からの個人プロセス記述の導出」，情処研報，SE 90-17（1993）

(3) 松永，飯田，荻原，井上，鳥居：「図式表現を用いたソフトウェア構成・実行システムの試作」，信学論 D-I，Vol.J76-D-I No.6（1993）

(4) T.Shepard，S.Sibbald，C.Wortley：「A Visual Software Process Language」，CACM，Vol.35，No.4（1992）

(5) 佐伯元司：「ソフトウェアプロセスのモデル化へのネットの応用」，情報処理，Vol.34，No.6（1993）