解　説

IPS

ネット指向パラダイムを求めて

4. ソフトウェアプロセスのモデル化へのネットの応用†

佐　伯　元　司††

1. はじめに

ソフトウェア開発のプロセスをモデル化し，形式的に記述する研究が盛んに行われている[1]．ソフトウェアプロセスを形式的に記述することは，

1. 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，

2. プロセスに従って開発者をナビゲートすることにより，効率的なソフトウェア開発支援を行うことができる，

3. 検証や検査，シミュレーションなどの手法と組み合わせることにより，そのプロセスの特性を分析し，プロセスのマネージメントに役立てることができる[2]，またプロセス自身の欠点を見つけることができ，その改善に役立てることもできる，

といった点で有用と思われる．Osterweil も述べているように[3]，ソフトウェアプロセスもソフトウェアとみなすことができるため，形式的に記述する際には，ソフトウェアの仕様記述に用いられる手法がそのまま，もしくは拡張されて用いられる．特にソフトウェアプロセスは，並列システム，分散システム，実時間システム，通信システムのすべての特徴をあわせもっているため，これらのシステムの仕様化に用いられている手法が適用されることが多い．記述が実行できる，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でソフトウェアプロセスをモデル化する研究も盛んである．本稿では，これらのネットを用いて，ソフトウェアプロセスのモデル化を行う手法を概説し，その利点や問題点について議論する．ネットの適用可能性に関する検討を行うため，ソフトウェアプロセスの共通問題を設定し，その問題を種々のネットで記述し，評価を行う．

2. では本稿で扱うネットの特徴を述べ，**3.** でソフトウェアプロセスのモデル化手法について述べる．種々のネットでプロセスを記述し，比較・検討を行った結果について，**4.**，**5.**，**6.** で述べる．ソフトウェアプロセスの特徴の一つに，ソフトウェアプロセス記述自身を作り出すようなメタプロセスの存在がある．このようなメタプロセスの記述については，**7.** で議論する．

2. ネットとは

ネットは，一般的にはその構造と実行規則とから構成される．ネットの構造は，数学的なグラフ，つまり 1)ノード(節)の集合，2)エッジ(辺)の集合，3)どのノードがどのエッジで接続されているかという連結関係，によって表現される．連結関係には数学的な関数や関係 (Relation) で記述するテキスト的表現と，図的表現とがある．ソフトウェア工学の分野では，了解性の良さから，図的表現をユーザインタフェースとして用いている場合が多い．このネットの構造は，静的であり，記述対象システムの動作などに応じて変化する．たとえばノードが生成されたり，連結関係が動的に変化したりすることはない．ネットによっては，複数種類のノードやエッジをもつものもある．

対象システムの静的な構造だけでなく，その時間変化をネットで表現するために，状態概念をもつネットがある．たとえば，ペトリネットではノード（プレース）にトークンを置くことができ，現在どのノードに何個トークンが置かれているか（マーキング）で，現在の状態を表す．このようなネットは，初期状態が与えられると，そのネットの構造とネット固有の実行規則に従って，

† Applying Net Theory to Software Process Modeling by Motoshi SAEKI (Dept. of Electric & Electronic Engineering, Faculty of Engineering, Tokyo Institute of Technology).

†† 東京工業大学工学部電気電子工学科

表-1 プロセス記述に用いるネットとプロセスの側面への適合性

ネット	記述対象	機能面	動作面	構成面
データフロー図[7]	アクティビティ，ファイル，入出力プロダクト	○		
SADT[8]	アクティビティ，入出力プロダクト，資源割当て	○		△
システム仕様図[28]	アクティビティ，入出力プロダクト（ストリーム結合，状態ベクトル結合），動作タイミング，同期	○	△	
Gantt チャート	アクティビティの実行例，実行開始・終了時刻		○	
実体構造図[28]	アクティビティや資源の動作順序・時刻		○	△
拡張フローチャート	アクティビティの実行順序		○	
R-Net[12]	アクティビティの実行順序，アクティビティ間の通信，ファイル	△	○	
状態遷移図[10], State Chart[11]	アクティビティの状態遷移		○	
ペトリネット	アクティビティの実行順序，資源割当て	△	○	△
実体関連図[17]	資源，資源間の関係			○
オブジェクト間通信図[18]	資源，資源間の通信			○
Role Interaction Net[1]	資源，資源間の通信，通信の順序		△	○
Transformational Schema[29]	アクティビティ，ファイル，入出力プロダクト，アクティビティの状態遷移	○	○	
Statemate[11]	アクティビティ，その状態遷移，ファイル，入出力プロダクト，資源間の通信	○	○	○

○：適　△：部分的に適

状態が遷移していく．これがネットの実行である．一般的には，このような実行メカニズムまでもったものをネットと呼んでいるが，実体関連図(Entity Relationship Diagram) のように実行メカニズムをもっていないものもソフトウェア工学の分野で多く利用されていることを考慮し，本稿ではこれら静的な構造のみしかもっていないものも含めて考える．**表-1** にソフトウェアプロセスに適用可能と思われる代表的なネットをあげる．

3. プロセスのモデル化手法

3.1 モデル化のための三つの側面

ソフトウェアプロセスを記述するために，ソフトウェアシステムと同様に，以下の一般的な三つの側面からとらえる手法がある[4),5)]．

1. 機能的側面 (Functional Perspective)
プロセス中にどのようなアクティビティ（作業）があるか，そのアクティビティによってどのようなプロダクトが生成されるか．

2. 動作的側面 (Behavioral Perspective)
プロセス中のアクティビティの実行順序などの時間的な振舞いはどうなっているか．

3. 構成的側面 (Organizational Perspective)
アクティビティを実際に実行する資源（たとえば人間やコンピュータツールなど）は何か．

ソフトウェアシステムの形式的仕様記述には，種々のネットが用いられている．それらはソフトウェアシステムのある側面をとらえるのには適しているが，別の側面を記述するには適していないことがある．たとえば，実体関連図は対象システムの動作的な側面を記述するのには適切ではない．各ネットの特性を十分理解し，適切な側面の記述に適用することが重要である．ネットをソフトウェアプロセスのモデル化に適用する際も同様である．Kellner は，Statemate の三つのチャート図 (Activity Chart, State Chart, Module Chart) を用いて，三つの側面から記述を行った[5)]．これは，各ネットの特性をうまく活かした例であろう．これまで，ソフトウェアプロセスを記述する手法は，さまざまなものが提案されてきた．文献 1) でそれらが詳説されているので，そちらを参照されたい．

3.2 メタプロセス

ソフトウェアプロセスが通常のソフトウェアと異なる点の一つは，実施されるソフトウェアプロセス自身をプロダクトとするメタプロセスの存在である．たとえば，ソフトウェア開発がスタートする時点を考えてみよう．まず最初に，開発責任者は作業計画を立て，作業者と作業分担を決める作業を行うであろう．この作業の出力は，それ以降で実施されるプロセスを記述したものである．このような作業過程は，メタレベルのプロセスである．このメタプロセスのモデル化については，どのネットを用いても，共通の困難性があるの

で，**7.** で個別に議論する．

3.3 例 題

次章以降で使用する例題として，すでに開発が終わったソフトウェアの要求 (Requirements) が変更されたとき，どのようにしてソフトウェアの修正を行っていくかというプロセスの一部を考えてみよう．この例題は，文献 6) のものを単純化したものである．

アクティビティは，Modify Design, Review Design, Modify Code, Test Unit, Monitor Progress の5つがある．Modify Design は，現在の設計仕様書 (current design) を取り出し，要求変更 (requirements change) に従って修正する．修正した設計仕様書 (modified design) は，Review Design, Modify Code に送られ，おのおの仕様書のレビュー，コードの修正が行われる．レビュー結果 (review document) は，Modify Design に送り返され，その結果によって再修正を行ったり，承認された (approved) 場合は何も行わなかったりする．Modify Code によって修正されたソースコード (modified code) は，Test Unit でテストがなされる．テストにパスすれば，全体の作業は終了，そうでなければテスト結果を Modify Code に返し，デバッグや再修正を行う．おのおののアクティビティが終了するたびに，その notification が Monitor Progress に送られ，そこで作業の進捗状況がチェックされる．作業の遅れが重大な場合 (severe) は，作業全体の中止 (recommendation to cancel) を CCB (Configuration Control Board) に勧告する．CCB によって中止が決定されると，Monitor Progress は，ほかのアクティビティにそれを通知 (cancellation) する．遅れが軽度な場合は，その遅れに合わせて作業計画を変更 (updated project plan) し，対処する．

この例題には Project Manager, Design Engineer, Quality Assurrance Engineer (以下，QA Engineer と略す) と2人の Engineer の合計5人の作業者がいる．おのおのの作業の割当ては，**表-2** のようになっている．また，使用されるツールは，コンピュータ (コンパイラとテスト実行用ツールをもっている)，コンピュータファイル (Source Code, Test Package を格納しておく)，紙ファイル (Design Document をファイリングする) といったものがある．

表-2 作業の割当て

アクティビティ	作業担当者
Monitor Progress	Project Manager
Modify Design	Design Engineer
Modify Code	Design Engineer
Review Design	Design Engineer, Engineer #1, Engineer #2, QA Engineer
Test Unit	Design Engineer, QA Engineer

4. プロセスの機能的側面

例題を機能的な側面，つまり例題プロセスにどのようなアクティビティがあり，それらの入出力プロダクトにはどのようなものがあるかを考えてみる．以下の節では，機能的側面を表すのに適した代表的なネットであるデータフロー図[7] と SADT[8] を用いた記述例を紹介する．

4.1 データフロー図

データフロー図[7] は，ソフトウェアシステム中のデータの流れを記述するためのもので，データの流れを表す矢印，データを変換するバブル (プロセスとも呼ばれる)，データを保存するデータストアより構成される．これら以外にもデータ構造を記述するデータ辞書，プリミティブな操作内容を記述するミニスペックがある．バブルは，さらに階層的にバブル，データフロー，データストアに分解することができる．例題プロセスをデータフロー図で書いたものを**図-1** に示す．たとえば，Modify Design アクティビティの入力は current design (Design Document データベースより)，review document (Review Design より) であり，modified design (修正済み設計仕様書) を Review Design と Modify Code アクティビティに出力する．

4.2 S A D T

SADT もデータフロー図と同様，アクティビティとその入出力プロダクトの関係を記述したものである．**図-2** に例題プロセスを記述した例を示す．データフロー図のような機能 (バブル) とファイル (データストア) の区別はもっていないかわりに，フロー (矢印) の長方形に接続されている位置によって，入出力の役割が区別される．たとえば，Monitor Progress は notification によって実行開始が制御され，Project Manager によって実行される．このような機構により，不十分で

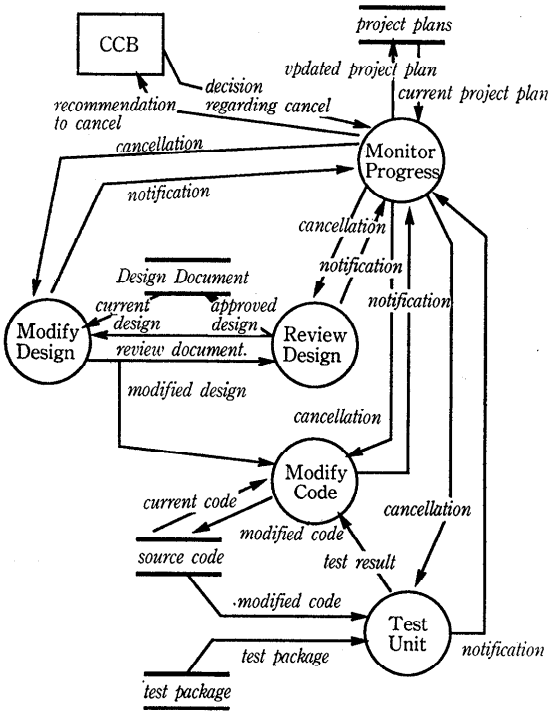


図-1 データフロー図による記述

はあるが，動作的側面や資源の割当てといった構成的側面も記述できるようになっている．

この記述例では長方形をアクティビティとしたが，プロダクトやデータを長方形に割り当ててもよい．この場合，入力フローによってそのデータを生成するアクティビティを，出力フローによってデータを使用するアクティビティを表現する．

このように，SADT では図形要素の使い方が1通りだけに規定されているのではなく，いくつかの使い方ができるようになっている．

5. プロセスの動作的側面

機能的側面では，プロセス中のアクティビティとその入出力プロダクトが何であるかということしか記述しておらず，どのような順序で，あるいはどのようなタイミングでアクティビティが実行されるかということには触れられていない．

例題中の各アクティビティの実行順序は，以下のようになる．まず，Modify Design が実行され，それが終了すると Review Design が実行される．Review Design の結果は，設計仕様書の承認 (approved)，もしくは仕様書の変更要請 (change) である．変更要請の場合は，Modify Design, Review Design が，仕様書が承認されるまで繰り返し実行される．Modify Code は，Review Design で設計仕様書が承認 (approved) されてから実行されるのが通常であるが，承認されていなくても作業者の判断で実行を始めても構わないものとする．つまり，修正済みの設計仕様書を作業者が受け取り次第，作業を開始することができる．Modify Code が終ると，Test Unit が実行され，修正されたソースコードのテストが行われる．テストに失敗する (failure) と，再び，Modify Code, Test Unit がこの順で実行される．このサイクルは，テストにパス (pass) するまで，繰り返し行われ

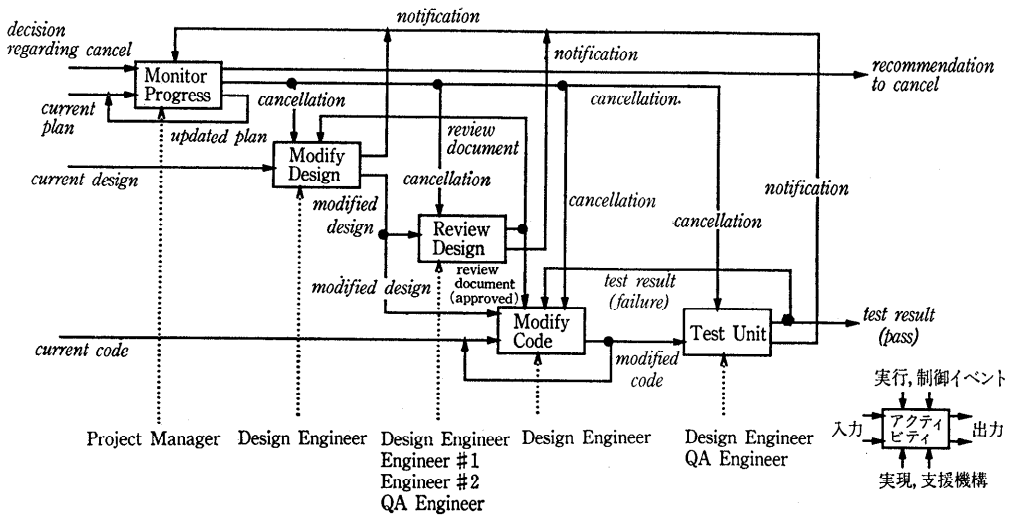


図-2 SADT による記述

表-3　ネットによる動作的側面の記述の比較

ネット	記述力							実行可能性解析可能性
	分岐	繰返し	並列	非決定性	到達性	時刻	割込み	
動作指向型 (Action Oriented)								
Gantt チャート	×	×	○	×	○	○	×	×
拡張フローチャート	○	○	○	○	×	×	×	○
R-Net	○	○	○	△	△	△	△	○
状態指向型 (State Oriented)								
State Chart	○	○	△	○	×	○	△	○
ペトリネット	○	○	○	○	×	○*	△	○

○：そのまま記述できる
△：工夫すれば記述可能であるが複雑になったり，不完全であったりする
×：記述が困難
* Timed Petri Net を使用した場合

る．パスすれば，プロセス全体は終了する．Modify Design, Review Design, Modify Code, Test Unit の各アクティビティとは並列に，進捗状況をチェックする Monitor Progress が実行される．

次節以降では，動作的側面を記述するのに適した代表的なネットを用いて例題記述を行った例の一部について述べる．なお，**表-3** に例題プロセスの動作的側面を記述した際の比較をまとめてあげておく．

5.1 Gantt チャート

Gantt チャートは，プロセス中のアクティビティのタイムスケジュールをカレンダ形式で記述したものである．**図-3** にその例を示す．この例では，Review Design アクティビティ，Test Unit アクティビティとも1回ずつやり直しがあることを見込んだスケジュールになっている．

Gantt チャートは，アクティビティの動作順序を，自分より先に実行が始まるアクティビティ，後に始まるアクティビティといった関係で単純に記述しているのみである．したがって，アクティビティの任意回の繰り返しや非決定的な動作などは表現できない．しかし，一見しただけで，プロセス中のアクティビティの時間的な関係の概略が分かるという利点がある．

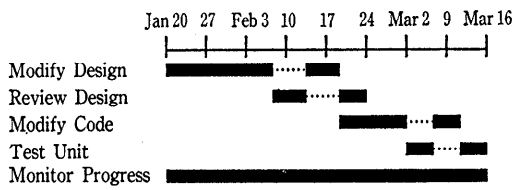


図-3　Gantt チャートによる記述

5.2 拡張フローチャート

フローチャートは，従来から逐次プログラムの制御を抽象化して記述するために用いられてきた．フローチャートの分岐節に AND 節や OR 節の概念を導入し，並列に動作したり，非決定的に動作するプログラムやシステムを記述できるように拡張が施されているものもある．この拡張フローチャートで例題を記述したものを**図-4** に示

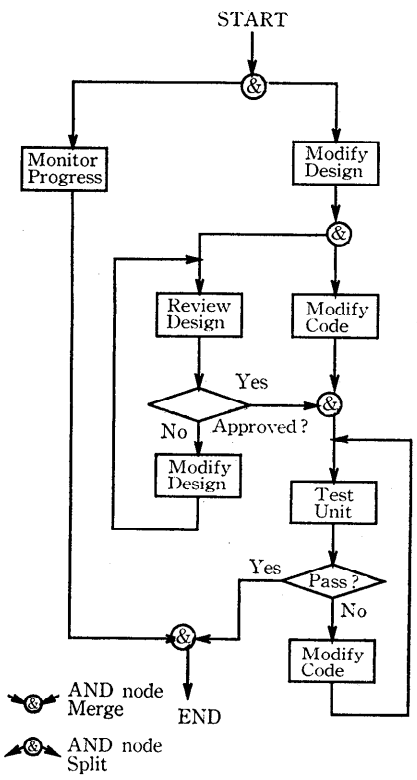


図-4　拡張フローチャートによる記述

す．図中の & で示された節が AND 節で，この箇所で分岐したフローは並列に実行されることを表す．たとえば，一番上の AND 節で分岐した二つの処理 Monitor Progress と Modify Design は並列に実行が始められることが記述されている．このような拡張フローチャートを用いたものに，VPL[9] がある．

フローチャートは，Gantt チャートと同様に一見しただけで動作の順序を容易に理解することができるが，その反面，複雑な動作を記述することが難しい．たとえば，作業の遅れが重大で，Monitor Progress が実行中の全アクティビティを cancel し，動作の流れを変更するといった，一種の割込みによる動作の流れを記述することが難しい．実際，図-4 には．この cancellation に関する動作は記述されていない．

5.3 状態遷移図 (State Chart)

状態遷移図は，通信プロトコルや実時間システムなど動作のタイミングが仕様の本質となるようなソフトウェアシステムの仕様記述に伝統的に広く用いられている[10]．複雑なシステムを記述する際には，状態数が膨大なものになってしまうという問題があるが，State Chart[11] などのように状態の階層的分割を行うことによって，記述の上での状態数を減らし，読みやすくする手法もある．**図-5** に State Chart を用いて，例題の動作的側面を記述した例を示す．Idle状態，Design Approved 状態，Wait for decision (CCB からの決定を待っている状態)，Stop 状態，END 状態を除く状態は，状態名となっているそのアクティビティが実行中であることを表す．Modify Code の実行状態については，すでに修正された設計仕様書が承認されている状態 (with Approval) とそうでない状態 (without Approval) とに分けた．どのようなときに状態遷移が起こるか，状態遷移が起こるときにどのような動作が生じるかは矢印ごとに記述される．たとえば，A/B と記述されていれば，イベントAが生じる，もしくは述語Aが真となればその遷移が起こり，イベントBが起こることを表す．

全体は，実線で区切られた二つの状態遷移機械 (Modify Cycle と Monitoring) で構成され，これらは並列に動作する．Modify Cycle は，さらに並列に動作する二つの機械 Review Cycle, Test Cycle に分割されている．つまり，合計三つの機械が並列に同期をとりながら動作する．Modify Design アクティビティが終了すると，イベント end_design が生じる．このイベントが生じたことによって，Test Cycle 中で Idle から Idle への遷移あるいは Modify Code without Approval への遷移と，Monitoring 中で Idle から Monitor Progress への遷移が同期して起こる．前者の遷移のうちどちらが起こるかは非決定的である．各アクティビティの実行終了後，Monitor Progress の実行状態となる．その実行の結果，アクティビティの遅れが重大であるとなると，イベント severe を起こして Wait for decision へ遷移する．CCB の決定が中止 (decision regarding cancellation) であると，cancellation というイベントを出力して遷移する．このときほかの二つの遷移機械もどんな状態にあっても cancel 状態へと遷移する．この例では，三つの状態遷移機械が同期をとりながら並列に動作するように記述されているが，数学的には真の並列動作の意味づけがなされているわけではなく，三つの機械の状態遷移がインタリーブして動作するように意味づけられている．

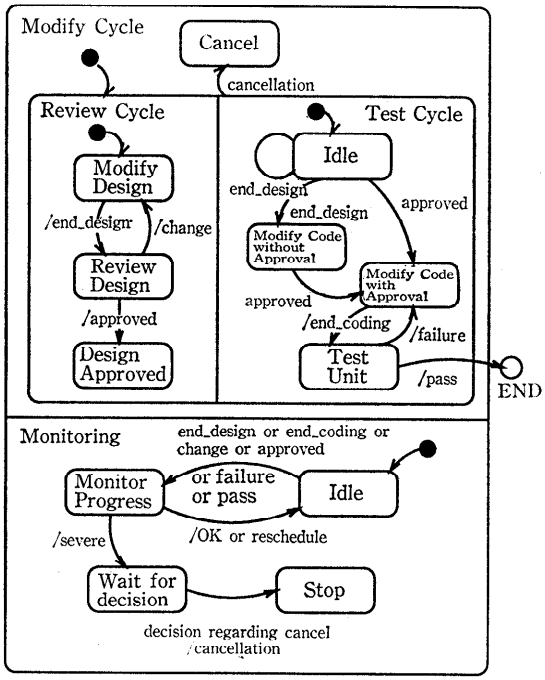


図-5 状態遷移図による記述

5.4 R-Net

R-Net は，アメリカの TRW 社によって開発された実時間システムの要求定義手法 SREM[12]

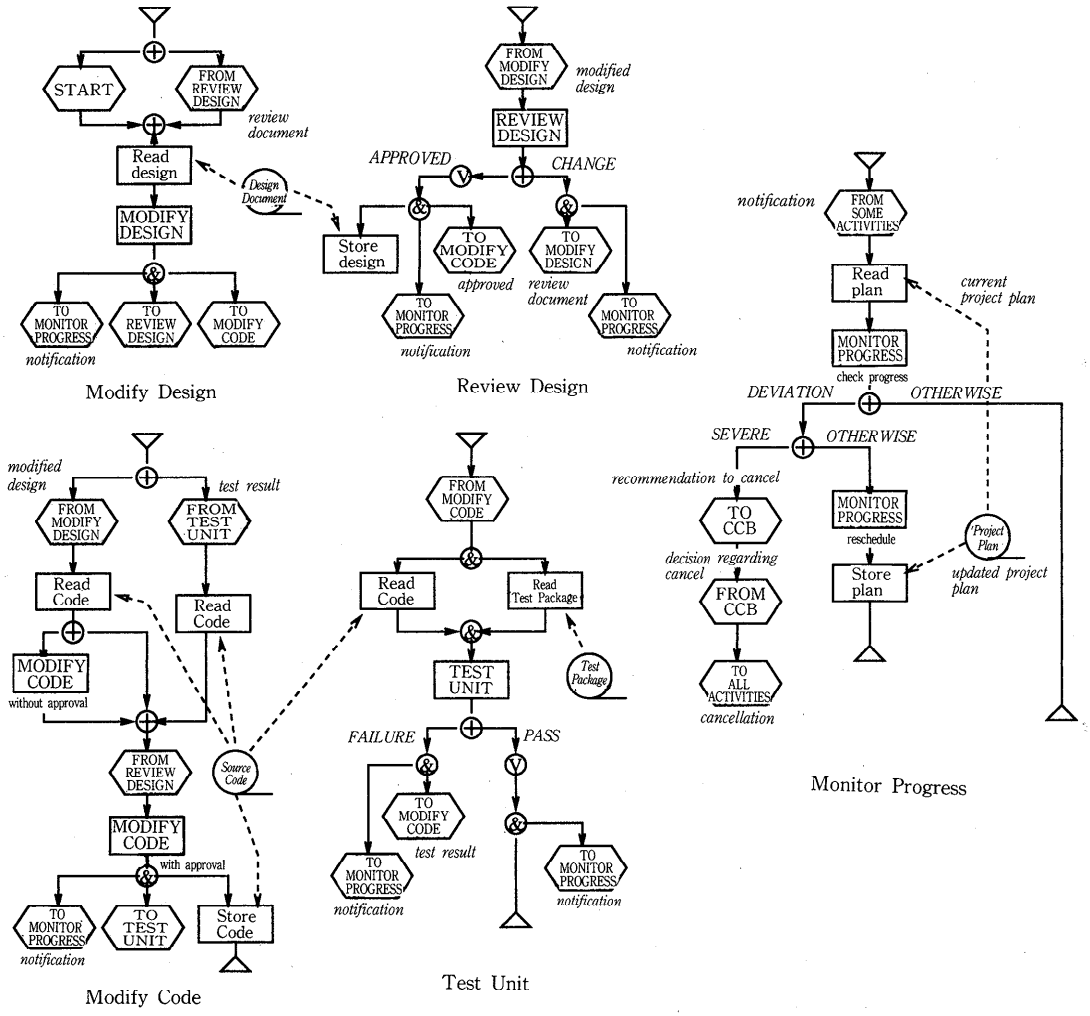


図-6　R-Net による記述

の中で用いられた一種のフローチャートである．SREM では，互いにメッセージ通信を行いながら並列に動作する複数の R-Net でシステムを記述する．一つの R-Net はシステムの一つの機能を表し，ほかの R-Net からのメッセージ受信によって実行が始まる．各 R-Net は，入力メッセージを受け取ってから出力メッセージを発するまでの動作(データ計算やデータベースへの保存など)を記述したものである．さらに，R-Net 中のパス上の流れをチェックできる確認点(Validation Point)と呼ばれる特殊な節を書くことができ，これによりシステムの性能要求（たとえば，その点に到達するまでの時間など）を記述することもできる．

図-6 に R-Net による例題の記述例を表す．例題プロセスの5つのアクティビティをそのまま分割された機能とし，おのおのの内部の動作を一つの R-Net で記述した．図中の長方形が処理手続きを，六角形がほかの機能（外界）とのメッセージの入出力処理を表し，これらは矢印の方向に従って実行が行われる．& 印ノード（AND 節）で分岐している処理手続は並列に実行され，＋印ノード（OR 節）は処理の分岐を表す．Vの書かれたノードが確認点で，modified document が approved になる箇所と modified code が Test Unit によって pass される箇所に挿入されている．これによって，modified design はいつかは approved にならなければならないといったことを間接的に表現することはできる．

R-Net は，並列実行や外部とのインタラクションの記述が行えるとはいえ，**5.2** で述べたフロー

チャート同様，単純な動作系列しか表現できない．したがって，図-6 では Monitor Progress からの cancellation メッセージによる割込み処理が記述できていない．R-Net を用いるには，記述対象を，各機能の動作が R-Net で記述できるほど十分 Primitive な機能に分割しなければならない．そのため，大規模システムでは分割された機能の数が多くなったり，分割単位がわれわれの直観と合わなくなったりする可能性がある．

通信システムの記述に用いられる SDL[30] も同様な言語／ネットで，プロセス記述にも適用可能である．

5.5 ペトリネット

ペトリネット[13]は，プレース (place)，トランジション (transition) と呼ばれる二種類のノードと有向矢印から構成される二分有向グラフの構造をしている．プレースにはトークンと呼ばれるマーカを置くことができ，このトークンがある規則に従ってプレースを移動していく．このトークンの移動によって，ソフトウェアシステムの動作を表現する．トランジションは，入力となっているプレースすべてに一つ以上のトークンが置かれているとき発火可能になり，出力となっているプレースにトークンを移動させることができる．

例題記述例を図-7 に示す．プロセス中のアクティビティはプレースで表現し，アクティビティを表すプレースにトークンが置かれると，そのアクティビティは実行中であることを示す．図中の丸矩形 (round rectangle) がアクティビティを表すプレースで，それ以外のプレースは丸で表した．また，図中のトークンの配置は初期マーキングを表している．START トランジションが発火すると，トークンが Modify Design と Monitor Progress プレースに置かれ，これらのアクティビティが並列に実行される．Modify Design が終了すると，同様にして Review Design の実行が始められる．Modify Code アクティビティがただちに実行されるか，それとも遅れて実行が開始されるかは非決定的に選択される．Review Design の結果，修正された仕様書は change が必要かもしくはそのま

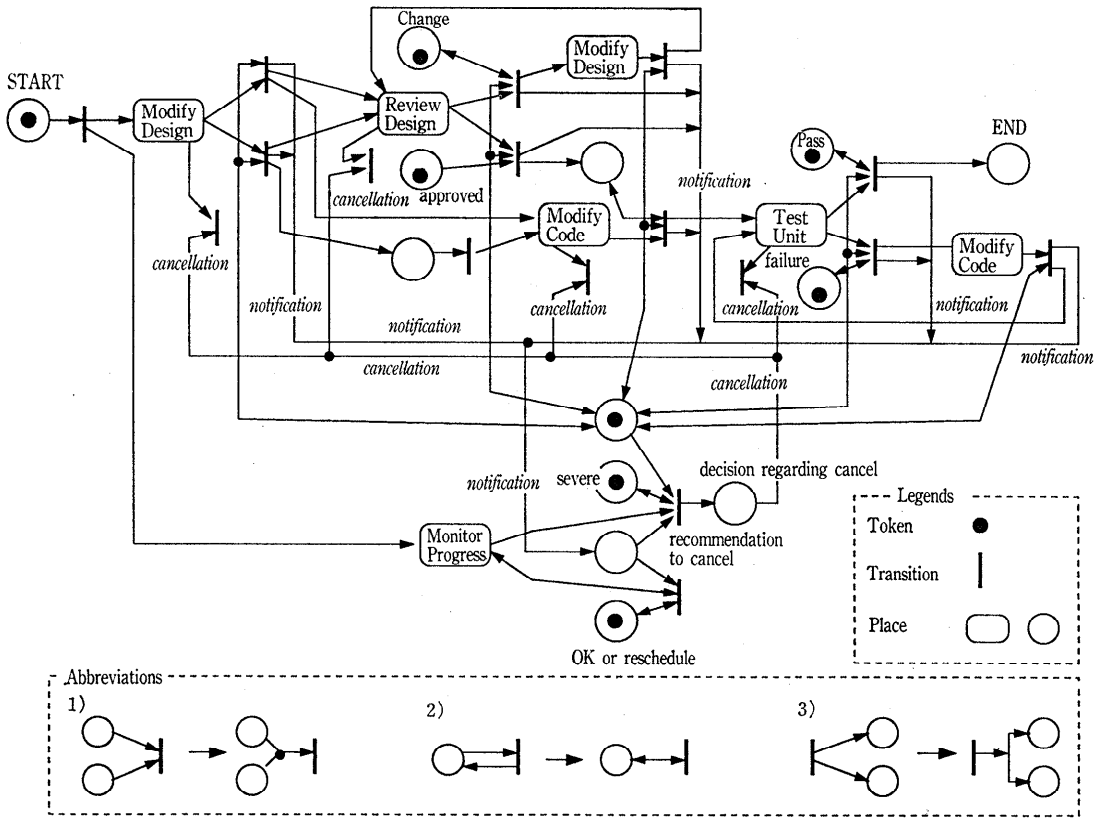


図-7 ペトリネットによる記述

ま承認される(approved)されるかのどちらかである．この条件分岐は，条件を表すプレースに置かれているトークンの移動によって行われる．

ほかのネットと比較すると，ペトリネットは動作順序に関する種々の制約を細部まで記述することができる．たとえば，「アクティビティの cancellation は次のアクティビティへの遷移より優先する」という非決定的動作の優先順位も表現できる．これは用意されている記述のための要素が Primitive であることによるが，その反面，単純な動作の記述でもネットが複雑になってしまう可能性がある．たとえば，トランジションの発火条件が「入力プレースのすべてにトークンが置かれている」という AND 型の条件となっているため，OR 型の条件による動作を記述するには工夫が必要である．また，「Review Design の結果，いつかは必ず modified design は承認 (approved) される」といった到達性を書くのは難しい．これは，フローチャート，状態遷移図でも同様である．

ここでの例は，動作的側面のみを記述したが，プレースやトークンの種類を増やし，ほかの側面も合わせて了解性よく記述できるようにしたペトリネットに，DesignNet[14] や FUNSOFT Net[15] がある．しかし，記述対象プロセスが複雑になってくると，ネットが複雑になり，ネットの動作を追跡しにくくなり，了解性が悪くなる．

6. プロセスの構成的側面

ソフトウェアプロセスの構成的側面とは，そのプロセスを実行するための資源やその間の関係からとらえた側面である．資源には，プロセスを実際に実行する作業者や使用するコンピュータシステム，開発用ツールといったものから，ドキュメントを蓄積しておく紙ファイル (Paper File) や書庫といったものも含まれる．資源間の関係とは，作業者間関係，作業者の組織構成，作業者間のコミュニケーションやコンピュータネットワーク構成といったものが含まれる．

6.1 実体関連図

実体関連モデル (Entity Relationship Model)[16] は，対象世界の構造を“もの：実体”(entity) とその間の“関連”(relationship) といった二種類の概念によって表現するためのモデルで，図として記述したものが実体関連図 (Entity Relationship Diagram) である．

例題の構成的側面を実体関連図で記述した例を**図-8** に示す．丸矩形が実体の集合(実体クラス)，実線がその間の関連を表す．たとえば実体クラス Worker のインスタンスは，Computer のインスタンスと accessible という関連をもつ．実線の両端に付加されている数字は，関連の基数 (Cardinality) を表している．たとえば，Project と Engineer の関連 work-for は，1 対 2 対応の関連であることが規定されている．これは，例題では Engineer が 2 人割り当てられていることによる．実体がもっている属性 (attribute) は丸矩形の中に書かれている．

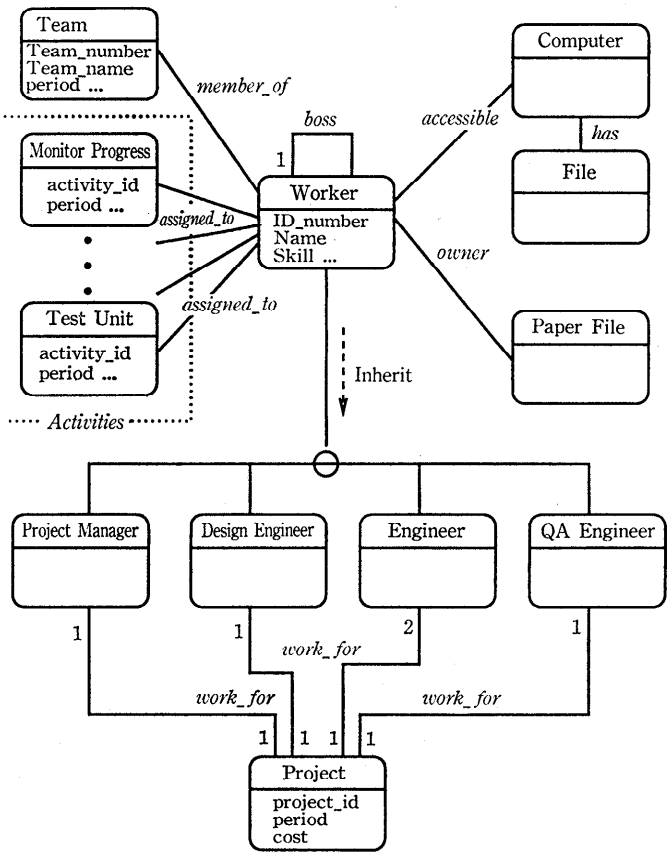


図-8 実体関連図による記述

たとえば，Worker はその属性として，ID-number (識別番号)，Name (氏名)，Skill (熟練度) などをもっている．記述を簡単にするために，Coad らのオブジェクト指向分析法[17]で取り入れられている継承 (Inheritance) 機構を用いた．実体 Worker がもっている属性や関連は，そのサブクラスである Project Manager, Design Engineer, Engineer, QA Engineer に継承される．たとえば，Project Manager は，ID-number, Name, Skill などの属性をもち，Team, Computer, Paper File などと member-of, accessible, owner といった関連をもつ．この継承機構を用いることにより，同じ記述を何回もする必要がなくなり，記述量を減らすことができる．

6.2 オブジェクト間通信図

実体関連図が組織などの資源間の静的な構造を表すのに対し，オブジェクト間通信図 (Object Communication Diagram)[18] は，資源間の通信といった動作に関連した構造を表現するのに用いられる．**図-9** に例題をオブジェクト間通信図で記述した例を示す．この例題では，作業者間の通信手段には電子メール (E-mail)，手渡し (Hand-carried)，口頭 (Verbal) がある．これらの手段によってプロダクトが作業者間に流れていく．たとえば，Monitor Progress アクティビティを実行した Project Manager は，各アクティビティの作業担当者より作業終了の notification を電子メール (E-mail) によって受けとり，同様に電子メールによって各作業者に Cancellation を知らせる．

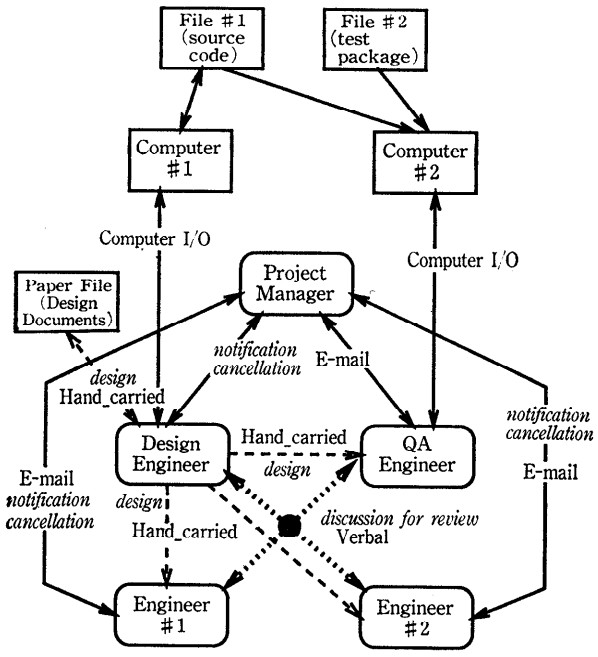


図-9 オブジェクト間通信図による記述

6.3 Role Interaction Net

Role Interaction Net は，同様に資源間の通信を表現するネットであるが，オブジェクト間通信図に対して，メッセージシーケンスチャート (MSC) のように，通信の時間的系列が記述できるように拡張されている．ひとまとまりの通信を一つのネットで表現し，通信の起こる順序に従ってこれらを表現したネットを並べて配置する．通信を表すエッジにはその通信に関連した動作を付加する．**図-10** に Review Design 部分の Role Interaction Net の記述例を示す．Design Engineer は，修正した仕様書をまず QA Engineer に送る (Send modified design)．QA Engineer は仕様書を受けとると，それを Review チームの各メンバに配布する (Distribute modified design)．メンバからの review を集める (Collect Reviews) のも QA Engineer の仕事である．つまり，QA Engineer が Review チーム内で責任者的な役割を果たしている．このように，Role Interaction Net では作業者が通信という観点からチーム内でどのような役割を果たしているかといったことも表現できる．

7. メタプロセス

アクティビティのスケジュール作成，作業者や資源の割当て作成，プロセスの進捗状況に応じたアクティビティ変更やそのスケジュールの変更，プロセス自身の改善といったアクティビティは，メタレベルのアクティビティである．通常のアクティビティと異なる点は，プロセス記述を出力し，出力されたプロセス記述に従って，以降の実行が進められていく点である．このようなアクティビティは，その作成・変更・改善対象となっているプロセス中に存在するため，結果的には，自分自身の実行を制御し，変えていく機構をもっていることになる．このようなメタレベルの機構を自己反映計算機構 (Reflection)[19] を用いてモデル

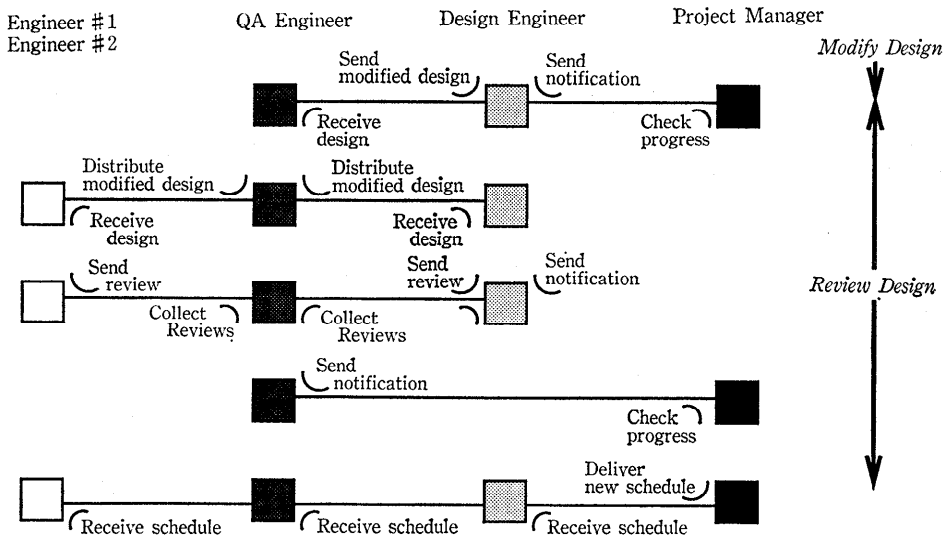


図-10 Role Interaction Net による記述

化した研究もある[20].

本章では，ネットでこのようなメタプロセスをどのようにモデル化すればよいかを考えてみよう．ネットの実行はその構造と実行規則によって規定されており，実行中に構造や規則が変化することはない．したがって，メタプロセスを記述するためには，これらを実行結果に応じて変化させる機構が必要である．本稿で使用した例題にはこのような機構を必要とするメタプロセスは含まれていなかった．ペトリネットを例にとると，メタプロセスを記述するために以下のような二つの手法が考えられる．

1. ペトリネットの構造を変える機構
2. 実行規則に従わない状態遷移を行う機構

1. の手法では，プレース，トランジションの生成・消去や，プレース，トランジションを接続しているエッジの生成・消去を行うオペレーションを用意し，これらのオペレーションを用いて，ペトリネットの構造を実行にともない，変化させていく．たとえば，あるプレースにトークンが置かれると，これらのオペレーションが発動され，ネットの構造が変化するように記述する．ペトリネットの状態はマーキングとその構造の対によって表され，マーキングはペトリネットの実行規則に従って遷移し，構造はこれらのオペレーションによって遷移していく．これらのオペレーションをメタオペレーションと呼ぶ．ペトリネットではないが，プロセスの実行を属性文法の解析木で表現し，部分木の張り替えや消去といったメタオペレーションにより，メタプロセスを表現した研究がある[21].

2. の手法では，ペトリネットのトークンの移動規則とは別に，あるプレースに自由にトークンを置いたり，消去したりするオペレーションを用意する．プレースにトークンを置くことによりプロセスの再実行ややり直しが，消去することによりプロセスの中止を表現することができる．前述のDesignNet[14] は，このような機構をもったネットである．

本格的にネットでメタプロセスを記述するためには，ネットの構造を変化させたり，ネットの実行規則の適用を制御したりする「メタネット」の研究が今後必要であろう．

8. おわりに

ネットをプロセス記述に用いることの利点の一つは，図的表現による了解性であろう．しかし，ネットのみで複雑なソフトウェアプロセスを完全に記述することは不可能で，これまでのソフトウェアプロセス記述への応用例は，ネットだけでなくほかのテキスト表現の言語を併用している[22),23].ソフトウェアプロセスを記述する目的によって，どの部分をどこまで記述するかは異なっており，またネットもどのレベルを記述するかによって適・不適がある．単純なプロジェクト管理のみに使用するのであれば，Gantt チャートのようなも

のでもよいであろう．また，エンジニアの教育[24]に使用するのであればプロセスの動作の大まかな流れを理解しやすい形で書かなければならない．高機能の言語やネットを一つ選んで使用するのではなく，目的に応じた使い分けが重要であろう．

本稿で述べたモデル化手法は，いずれもプロセス中心の考え方である．これに対し，プロセスによって生成されるプロダクトを中心に考える手法もある[25]．文献1)でも**3.**で述べた三つの見方に加えて，情報的な見方 (Informational Perspective) が議論されている．これは，プロダクトの構造やプロダクト間の関係をとらえる見方である．中間生成物まで含めて種々のプロダクトを統一的にモデル化し，管理する手法は，重要な研究課題の一つであり，ネット理論の成果が期待される．

ソフトウェアプロセスをモデル化／記述する際の難点は，人間の振舞いが関与している点である．特に，現実のソフトウェア開発は，複数の立場の異なる人間の協調作業，協同作業，共同作業 (cooperation, coordination, collaboration) からなっている．これらの作業の特質を明らかにし，モデル化していくことが，ソフトウェアプロセス記述を実際の開発支援に役立てる上で重要な要因となるであろう[26),27)]．

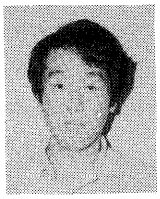
本稿は，1992年4月の「ネット指向ソフトウェア設計技術に関するチュートリアル」での原稿を加筆修正したものである．翁長教授(琉球大)，本位田氏(東芝)をはじめ，ご意見をいただいた諸氏に感謝いたします．

参 考 文 献

1) Curtis, B., Kellner, M. and Over, J.: Process Modeling, *Commun. ACM*, Vol. 35, No. 9, pp. 75-90 (1992).
2) Kellner, M. I.: Software Process Modeling Support for Management Planning and Control, In *Proc. of 1st International Conference on the Software Process*, pp. 8-28 (1991).
3) Osterweil, L.: Software Processes Are Software Too, In *Proc. of 9th ICSE*, pp. 2-13 (1987).
4) Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M. and Shtul-Trauring, A.: STATEMATE: A Working Environment for the Development of Complex Reactive Systems, In *Proc. of 10th ICSE*, pp. 396-406 (1988).
5) Kellner, M. I. and Hansen, G. A.: Software Process Modeling: A Case Study, In *Proc. of 22nd Hawaii Conferrence*, pp. 175-188 (1989).
6) Kellner, M., Feiler, P., Finkelstein, A., Katayama, T., Osterweil, L., Penedo, M. and Rombach, H.: Software Process Modeling Example Problem, In *Proc. of 1st International Conference on the Software Process*, pp. 176-186 (1991).
7) DeMarco, T.: *Structured Analysis and System Specification*, Yourdon Press (1978).
8) Ross, D. T. and Schoman Jr., K. E.: Structured Analysis for Requirement Definition, *IEEE Trans. on Software Engineering*, Vol. 3, No. 1, pp. 6-15 (1977).
9) Shepard, T., Sibbald, S. and Wortley, C.: A Visual Software Process Language, *Commun. ACM*, Vol. 35, No. 4, pp. 37-44 (1992).
10) Dasarathy, B.: Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them, *IEEE Trans. Softw. Eng.*, Vol. 11, No. 4, pp. 80-86 (1985).
11) Harel, D.: STATECHARTS: A Visual Formalism for Complex Systems, *Science of Computer Programming*, Vol. 8, pp. 231-274 (1987).
12) Alford, M. W.: A Requirements Engineering Methodology for Real-Time Processing Requirements, *IEEE Trans. on Software Engineering*, Vol. 3, No. 1, pp. 60-69 (1977).
13) Peterson, J. L.: *Petri Net Theory and the Modeling of Systems*, Prentice Hall (1981).
14) Liu, L. and Horowitz, E.: A Formal Model for Software Project Management, *IEEE Trans. on Software Engineering*, Vol. 15, No. 10, pp. 1280-1293 (1989).
15) Emmerich, W. and Gruhn, V.: FUNSOFT Nets: A Petri-Net Based Software Process Modeling Language, In *Proc. of 6th International Workshop on Software Specification and Design*, pp. 175-184 (1991).
16) Chen, P.: Entity-Relationship Model: Towards a Unified View of Data, *ACM. Trans. on Database Systems*, Vol. 1, No. 1, pp. 9-36 (1976).
17) Coad, P. and Yourdon, E.: *Object-Oriented Analysis*, Prentice Hall (1990).
18) Shlaer, S. and Mellor, S. J.: An Object-Oriented Approach to Domain Analysis, *ACM SIGSOFT Software Engineering Notes*, Vol. 14, No. 5, pp. 66-77 (1989).
19) 菅野，田中：メタ推論とリフレクション，情報処理，Vol. 30, No. 6, pp. 694-705 (1989).
20) Katayama, T.: A Hierarchical and Functional Software Process Description and its Enaction, In *Proc. of the 11th ICSE*, pp. 343-352 (1989).
21) Suzuki, M. and Katayama, T.: Meta-Operations in the Process Model HFSP for the Dynamics and Flexibility of Software Processes, In *Proc. of 1st International Conference on the Software Process*, pp. 202-217 (1991).

22) Inoue, K., Ogihara, T., Kikuno, T. and Torii, K.: A Formal Adaption Method for Process Descriptions, In *Proc. of 11th ICSE*, pp. 145-153 (1989).
23) Saeki, M., Kaneko, T. and Sakamoto, M.: A Method for Software Process Modeling and Description Using LOTOS, In *Proc. of 1st International Conference on the Software Process*, pp. 90-104 (1991).
24) 望月，山内，片山，鈴木：ソフトウェアプロセス—実時間処理システムにおけるケーススタディ，情報処理学会ソフトウェア工学研究会，Vol. 71, pp. 139-143 (1990).
25) 鯵坂，松本：ソフトウェアエンジニアリング・データベース kyotodb の設計と実現，情報処理学会論文誌，Vol. 33, No. 11, pp. 1402-1413 (1992).
26) 落水：統合環境 Vela におけるデザインレビュー支援，情報処理学会ソフトウェア工学研究会，Vol. 77, pp. 25-30 (1991).
27) 垂水：グループウェアのソフトウェア開発への応用，情報処理，Vol. 33, No. 1, pp. 22-31 (1992).
28) Jackson, M. A.: *System Development*, Prentice-Hall (1983).
29) Ward, P.: The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing, *IEEE Trans. on Softw. Eng.*, Vol. 2, No. 12, pp. 198-210 (1986).
30) 若原：SDL 言語の特質と処理系の現状と動向，情報処理，Vol. 31, No. 1, pp. 23-34 (1990).

(平成5年1月26日受付)



佐伯 元司（正会員）

昭和31年生．昭和53年東京工業大学工学部電気電子工学科卒業．昭和58年同大学院情報工学専攻博士課程修了．工学博士．同年より東京工業大学工学部情報工学科助手．昭和63年同大学工学部電気電子工学科助教授．ソフトウェア工学，マンマシン・システムなどの研究に従事．