階層的プロセスモデルの提案と そのＪＳＤへの適用

井上　克郎　　野村　研仁　　稲田　良造　　菊野　亨　　鳥居　宏次

（大阪大学　基礎工学部　情報工学科）

要旨：　ソフトウェアの生産性や信頼性を向上させるために，ソフトウェアの開発過程を理解しやすい形で正確に記述することが重要になってきている．本報告では，階層的なプロセスのモデルに基づき，開発過程を形式的に記述することにより，ソフトウェア開発方法論の一つであるジャクソンシステム開発法（JSD）を定義する．プロセスの記述については，手続き的記述によるプロセス・プログラミングが最近提案された．一方，プロセスは，入力を用いて出力を生成する過程であるとみなすいわゆる動作的モデルの提案もある．本報告では，動作的モデルに基づき，前提条件，完了条件，及びサブプロセスの関係の三つの要素から構成されるモデルを用いる．

JSD による開発の過程に関する記述は，例を用いた説明などによって定義を類推させる形で行なわれていることが多い．従って，これらの記述から JSD の定義を正確にとらえることは容易ではない．本報告では，まず，プロセスのモデルに基づいて，JSD を自然語で記述し，次に，代数的言語 ASL/1 を用いて記述するという段階を経て，JSD の定義を試みた．JSD を定義する作業を通して，JSD の各過程の入出力の関係，および，各過程で行なうべきことを明らかにすることができた．

1．まえがき

ソフトウェアの開発過程を，なんらかの形式的な体系を用いて記述する研究が，近年盛んに行なわれている[1)-10)]．これらの代表的なものとして，プロセス・プログラミング[1)]，Kaiser らの研究[2)3)]，Williams の研究[4)]などがある．

プロセス・プログラミングは，ソフトウェアの開発過程を PASCAL 的な構文を用いて，手続き的に記述することを提案している．これに対し，Kaiser らや Williams は，開発過程を手続き的に記述するのではなく，動作的なモデルに基づいて記述している．

Kaiser らは，ソフトウェアの開発過程を”ツールの実行”の集合であるととらえ，各種のツールを起動するための前提条件，ツール起動方法及びツール起動後の完了条件の三つの要素からなる”ルール”を用いて開発過程を記述している．

Williams は，ソフトウェアの開発過程を”活動”の集合であるととらえている．ここで”活動”は，前提条件，完了条件，行為，メッセージの組で表わされる．このうち”行為”は，ソフトウェア開発上の作業の単位であり，ツールの起動の他に，人間が行なう作業も含む．

これらの研究では，開発過程の一部を記述した例はあるが，開発過程全体にわたって詳細に記述した例はない．

一般に，ソフトウェアの開発過程は，明確に記述されていないことが多い．従って，ソフトウェアの開発過程を正確かつ理解しやすい形で記述することが望まれる．

本報告では，後述するようなモデルに基づき，代数的仕様記述言語 ASL/1 を用いて，ソフトウェア開発方法論の一つであるジャクソンシステム開発法（JSD）[11)12)]を記述する．

ジャクソンシステム開発法は，仕様要求分析・ソフトウェアの設計・プログラミングなどを含むソフトウェアシステム開発方法論である．JSD の特徴として，
(1) 仕様化と実現を明確に区別していること，
(2) システムが持つ機能の記述や仕様化を手掛ける前に，システムが対象とする現実世界についての記述とそのモデル化をすることなどが挙げられる．

(2) は，システムに対する機能要求に比べ，システムが対象としている世界のモデルは変化する可能性が低く，従って，まず最初に現実世界のモデル化を行ない，モデルの構造をシステムの構造に反映することにより，機能要求の変化に柔軟に対応することができるシステムを作ることができるという考え方に基づいている．

また，JSD は開発過程を適切に分割して順序付けし，各過程の完了基準および生成物の記法を記述するという方針をとっている．

JSD が開発方法として有効であること，仕様要求分析からプログラミングまでの広い範囲を対象としていること，開発過程や生成物を比較的詳細に定めていることから，この方法論を記述の対象として選択した．

本報告では他の研究で提案されている動作的なプロセスモデルを変更し，プロセスの定義が簡明で階層的なモデルを用いる（表１）．

本報告で用いるモデルは，Williams のモデルと同じように，ソフトウェアの開発過程は，ツールの実行など機械的に行なうことができる事と，人間が行なわなければならない事の両方の要素からなるという前提に基づいている．

形式的には，プロセス PS は，三字組 <PRE,POST,REL> で表わされる．PRE は前提条件の集合，POST は完了条件の集合，REL はサブプロセスの関係の集合である．前提条件は，プロセスへの入力が満たすべき条件を表わす．完了条件は，プロセスの出力が満たすべき条件を表わす．入力や出力が複数個存在する時は，それぞれの入出力に関する条件が必要である．一つのプロセス（親プロセス）をいくつかのプロセス（サブプロセス）の集まりとして表わしたい時は，親プロセスとサブプロセス，ならびに，サブプロセス間の関係をプロセスの定義中に記述する．

Williams のモデルでもプロセスを階層的に表現することはできるが，ここでは，サブプロセスの関係を用いて，階層性を明示的に記述することにより，開発過程のトップダウン的な記述を容易にしている．

2．ソフトウェア開発過程の記述

2.1 ソフトウェア開発方法論の記述に関する問題点

ソフトウェアの生産性や信頼性を向上させることを目的として種々のソフトウェア開発方法論が提案されている[13][14]．

しかしながら，方法論自体をなんらかの抽象的な概念に基づいて形式的かつ詳細に記述した例はない．

方法論自体が，理解しやすい形で正確に記述されていなければ，方法論に従って誰もが同じようにソフトウェアの開発を行なうことが困難になる．

表１　プロセスモデル

提唱者 （提唱年）	G.E.Kaiser ら －MARVEL－ (1987)	L.G.Williams (1988)	本論文で用いる モデル
開発過程 の 構成単位	ツールの実行	ツールの実行 および 人間の作業	ツールの実行 および 人間の作業
モデルの 要素	前提条件 完了条件 実行方法	前提条件 完了条件 作業内容 メッセージ	前提条件 完了条件 サブプロセスの関係

本報告では，抽象的なモデルに基づき，また，形式的な記述言語を用いることにより，理解しやすく，かつ，厳密な方法論の記述を行なう．本報告では，このような記述を定義と呼ぶ．

本報告では，方法論の定義に用いる抽象的なモデルとして，動作的なソフトウェアプロセスモデルを用いる．

次節では，本報告で用いるモデルについて述べる．

2.2 ソフトウェアプロセスモデル

プロセスは一つ以上の入力を用いて一つ以上の出力を生成する過程のことをいう．

プロセスは抽象度に応じて任意の大きさに分割することができるが，どのように分割した場合でも必ず入力と出力を持つ．

形式的にはプロセスを，前提条件，完了条件，サブプロセスの関係の組で表現する．

ここで前提条件とは，入力が満たすべき条件に相当し，完了条件とは出力が満たすべき条件に相当する．入出力はそれぞれ複数個存在することがあるが，この場合それぞれに条件がある．

前提条件や完了条件は，それらに対応する入力や出力が属する記述言語（なんらかのクラスであると考えてもよい）を用いて表わす場合や，他の言語を用いて表わす場合，また，その両方を用いる場合もあろう．

一般に，一つのプロセス（親プロセス）は，サブプロセスの集合であると考えることができる．サブプロセスはそれ自身プロセスとして別に定義される．この場合，サブプロセス間の関係として，各サブプロセスの実行順序と入出力を指定する．サブプロセス間の関係を定義する際の注意としては，

(1) あるサブプロセスを実行するときは，そのサブプロセスの前提条件を満たす入力が存在していなければならない，

(2) サブプロセスへの入力は，その親プロセスの入力，同じ親を持つ他のサブプロセスの出力，プロセス開始以前に存在が前提とされている入力（ツールなど）のいずれかでなければならない

(3) ある親プロセスに対応するサブプロセスの実行がすべて終了した時，親プロセスの完了条件を満たすような出力が生成されていなければならない．一般に，最後に評価されたサブプロセスの完了条件と親プロセスの完了条件は一致しない事があるが，そのような場合でも，実行されたすべてのサブプロセスの完了条件を用いて親プロセスの完了条件が満たされるようになっていればよい．

プロセスは，機械的にその値を評価できるような形式で記述できない場合がある．我々は，必ずしもすべてのプロセスが機械的に実行される必要はないと考える．ソフトウェア開発作業を行なう際には，かなり人手に頼らなければならないことが多いのが現状である．従ってプロセスに関する記述の一部を人間が評価する（人間がプロセスを実行する）ことを許すものとする．

3. JSD の定義

3.1 定義とは

ソフトウェア開発方法論は，自然語を用いて説明的あるいは例示的に記述されていることが多い．

このような記述には，あいまいさや，不明確さなどがあり，記述を読んだだけでは，方法論にそった手続きや，各過程の完了基準，作成すべき生成物の定義などを必ずしも的確に認識できるとは限らない．

むしろ，各方法論にそった過程を繰り返し経験することにより認識できるようになると考えられる．

ソフトウェア開発方法論は，ソフトウェアの生産性や信頼性を向上させることを目的として提案されているのであるから，方法論自体が理解しやすい形で記述されていることが求められる．

この要求を達成するためには，少なくとも次の項目を満たすような記述方法が用いられなければならないと考えられる．

(1) 形式的な記述言語を用いること．

(2) 抽象的なモデルに基づいて記述すること．

次節では，上記の項目を満たす具体的な定義の記述方法について述べる．

3.2 JSD の定義の基本方針

JSD は，開発過程等をかなり詳細に定めているが，それらの記述は例示的なことが多く，正確に定義をすることは容易ではない．

そこで，

(1) まず自然語を用いて，JSD による開発過程を記述する，

(2) (1)で作成した記述を用いて，形式的な記述を行なう，

という二つの段階を経て行なうことにした．なお（1）で自然語による記述を行なうが，このときに，プロセスモデルに基づいて記述することにより，現実の開発過程の実現方法に依存せずに，本質を捉えて記述することができる．

また，開発過程の抽象度にあわせてプロセスを分割し定義する．JSD は，前述したように五つの段階に分割することができる．そこでまず，この五つの段階から構成される，JSD 全体の過程”JSD プロセス”の定義を行なった．次に，五つの段階のうち実現化段階を除く四つの段階を自然語を用いて記述した．実現化段階は，作成したい個々のシステムが稼動する環境に依存する部分が非常に多いので，今回は記述を行なわなかった．

最後に，自然語による定義記述のうち，エンティティ構造段階について代数的言語 ASL/1 を用いて形式的に記述した．

4．自然語による JSD の記述

JSD プロセスの自然語による定義記述を表2に示す．

表2では，JSD プロセスの入力と出力が属する言語（クラスと考えればよい）および前提条件と完了条件，ならびに，JSD プロセスのサブプロセスの関係を定義している．

例えば，入力である”ソフトウェアシステム要求仕様書”は，”仕様書記述言語”（一般には自然語と同等である）に属し，システムが対象としている問題の領域に関する記述などがなされていなければならないと定めている．

なお，この表で用いられている”仕様書記述言語”，”SSD”（システム仕様図），”構造図”は，いずれも JSD 特有の生成物を記述するための言語である．これらの言語についてはそれぞれ別に定義されているものとする．

表2　ＪＳＤプロセスの自然語による記述

＜入力＞
・ソフトウェアシステム要求仕様書：仕様書記述言語
＜出力＞
・システム実現図：ＳＩＤ（システム実現図記述言語）
・スケジューラプロセスの構造図の集合：
set of 構造図
・サブルーチンの構造図の集合：
set of 構造図
＜前提条件＞
(1) ソフトウェアシステム要求仕様書に，システムが対象とする世界に関する記述，システムの機能要求，システムが作動する環境に関する記述があること．
＜完了条件＞
(1) 要求仕様にそった出力が作成されていること．
(2) サブプロセスの関係に示されている手順を経て，各出力が作成されていること．
＜サブプロセスの順序＞
ＪＳＤは，５つのサブプロセス
・エンティティアクション段階
・エンティティ構造段階
・初期モデル段階
・機能段階
・実現化段階
を，逐次的に実行する．
＜各サブプロセスの入出力の受け渡し関係＞
(注) 括弧付きの入出力は必ずしも必要ではない．
＜エンティティアクション段階＞
＜入力＞
・ＪＳＤプロセスに入力されたソフトウェアシステム要求仕様書
＜出力＞
・エンティティリスト
・アクションリスト
・共通アクションリスト
・エンティティ候補リスト
・アクション候補リスト
＜エンティティ構造段階＞
＜入力＞
・エンティティアクション段階が出力したエンティティリストとアクションリスト．
・ＪＳＤプロセスに入力されたソフトウェアシステム要求仕様書のうちアクションの発生順序に関する記述
＜出力＞
・エンティティ構造図の集合
＜初期モデル段階＞
＜入力＞
⋮

”サブプロセスの実行順序”の定義において，JSD プロセスは，エンティティアクション段階，エンティティ構造段階，初期モデル段階，機能段階，実現化段階の五つの段階を逐次的に実行すると定めている．

また，”サブプロセスの入出力の受け渡し関係”では，JSD プロセスと各段階，ならびに，各段階間の入出力の受け渡しの関係を定めている．

この表によって，JSD プロセスの前提条件，完了条件，および，サブプロセスの関係が定義されている．

次に，JSD プロセスのサブプロセスの一つであるエンティティ構造段階の自然語による記述について述べる（表３）．

エンティティ構造段階の入力は，アクションリスト，エンティティリスト，アクションの順序仕様の三つで，それぞれ，”行動”の集合，”個体”の集合，”仕様書記述言語”に属する．

出力は，エンティティ構造図の集合で”個体構造図”の集合に属する．

エンティティ構造段階の前提条件は，入力”アクションの順序仕様”にアクションが発生する順序が定められていることである．他の二つの入力については，前述した JSD プロセスの定義中で，エンティティアクション段階の出力であると定められているので，ここで新たに定義すべき条件はない．

エンティティ構造段階の完了条件は四つに分かれているが，いずれも出力”エンティティ構造図の集合”が満たすべき条件である．

表３　エンティティ構造段階の自然語による記述

＜入力＞
・アクションリスト：set of 行動
・エンティティリスト：set of 個体
・アクションの順序仕様：仕様書記述言語
＜出力＞
・エンティティ構造図の集合：set of 個体構造図

＜前提条件＞
(1) “アクションの順序仕様”が，アクションの発生する順序を定めていること．
＜完了条件＞
(1) “エンティティリスト”に含まれているすべてのエンティティについて，そのエンティティの構造図が存在すること．また，それ以外の構造図が存在しないこと．
(2) “アクションリスト”に含まれているすべてのアクションについて，アクションの定義に記述されているエンティティに対応する構造図の中に，そのアクションが存在すること．またこれ以外の構造図にそのアクションが存在しないこと．
(3) エンティティ構造図に表わされているアクション間の順序関係が，“アクションの順序仕様”の記述と一致すること．
(4) “エンティティ構造図の集合”が (1) から (3) の条件を満たし，かつ，それぞれの要素が“極小構造”になっていること．

5．JSD の代数的記述

5.1　代数的仕様記述言語

本章では，代数的仕様記述言語 ASL/1[15] を用いて行なった，JSD プロセスおよびエンティティ構造段階の記述について述べる．

ASL/1 は，代数的仕様記述言語 ASL/*[15][16] における文法及び公理の記述法を具体的に定めた言語である．

一般に代数的言語は，意味の定義が簡明である，抽象的に高いレベルのものから低いレベルのものまで同じ枠組の中で記述することができる，検証を形式的に行なうことができるといった特徴を持つ．従って，さまざまな抽象度の作業を比較的厳密に行なえるという特徴を持ち，ソフトウェア開発過程を記述するのに適していると思われる．

また ASL/1 の部分言語である関数型言語 ASL/F の最適化コンパイラが存在している[17]ので，ASL/1 による記述を ASL/F に変換すれば，開発過程の記述を実行することができる．

ASL/1 で書かれた一つのテキストは，すでに定義してある仕様を取り込む include 文，各関数の引数と関数値の型を「関数名:第一引数の型,第二引数の型,...,第 n 引数の型 -> 関数値の型」の形で指定する関数の型指定，項の合同関係を「項１ == 項２」で表現する公理の集合からなる．なおこのうち関数の型指定は，ASL/1 における表現式を指定する生成規則の記述に対応する．

基底データ型としては，ブール（boolean）（基本関数として，&，＝等を用いる），文字列（string）（＝），集合（set）（∈，＝，

```
TEXT ＪＳＤプロセス;

INCLUDE primitives;
INCLUDE ＪＳＤ生成物;

ＪＳＤ:仕様書記述言語 ->
  [ＳＩＤ,set of 構造図,set of 構造図];

＃１．前提条件
  ＪＳＤプロセスの前提条件:仕様書記述言語
     -> boolean;
1-1:ＪＳＤプロセスの前提条件(spec) ==
    システムが対象とする世界に関する記述
    があるか？(spec)
      &
    システムの機能要求があるか？(spec)
      &
    システムが作動する環境に関する記述が
    あるか？(spec);

＃２．完了条件
2-1:要求仕様にそったプロダクトを生成した
か？:ＳＩＤ,set of 構造図,set of 構造図,
     仕様書記述言語
          -> boolean;

＃３．サブプロセスの関係
エンティティアクション段階:仕様書記述言語
 ->  [set of 個体,set of 行動,
      set of 共通行動,
      set of 個体,set of 行動];
              ⋮

3-1:ＪＳＤ(spec) ==
実現化段階(
  [機能段階(
    [初期モデル段階(
      [エンティティ構造段階(
[pr1(エンティティアクション段階(spec)),
 pr2(エンティティアクション段階(spec)),
 spec]),
 pr1(エンティティアクション段階(spec)),
 spec]),
 pr3(エンティティアクション段階(spec)),
 spec]),
 spec]);
```

図１　ＪＳＤプロセスの代数的記述

```
TEXT エンティティ構造段階;

INCLUDE primitives;
INCLUDE ＪＳＤ生成物;

エンティティ構造段階 :
[set of 個体,set of 行動,仕様書記述言語]
  -> set of 個体構造図;

＃１．前提条件
  エンティティ構造段階の前提条件
    :仕様書記述言語 -> boolean;
1-1:エンティティ構造段階の前提条件(spec)
 ==
アクションの順序に関する仕様か？(spec);

＃２．エンティティ構造図の集合の完了条件
すべてのエンティティの構造図があるか？
    :set of 個体構造図,set of 個体
       -> boolean;
2-1:すべてのエンティティの構造図があるか？
      (ent_diag_set,ent_list) ==
    構造図のエンティティ名の集合
      (ent_diag_set)
      =
    エンティティリストのエンティティ名の
    集合(ent_list);

構造図のエンティティ名の集合
    :set of 個体構造図 -> set of string;
2-2:name ∈ 構造図のエンティティ名の集合
          (ent_diag_set) ==
    指定したエンティティ名を持つ構造図の
    集合(ent_diag_set,name) != 空;

指定したエンティティ名を持つ構造図の集合
    : set of 個体構造図,string
      -> set of 個体構造図;
2-3:ent_diag ∈ 指定したエンティティ名を
持つ構造図の集合
      (ent_diag_set,name) ==
    ent_diag ∈ ent_diag_set
      &
    name = 根の名前(ent_diag);
              ⋮
              ⋮
```

図２　エンティティ構造段階の代数的記述

!=），リスト（list）のみを用いる．

ＪＳＤの生成物を記述するための言語としては，個体，行動，共通行動，構造図，ＳＳＤ（システム仕様図），ＳＩＤ（システム実現図），仕様書記述言語を用いる．

これらの仕様の記述は include 文によって取り込まれる．

5.2 JSD プロセスの代数的記述

本節では，JSD プロセスの代数的な記述について述べる．

JSD プロセスの代数的な記述を図１に示す．

テキスト中では，include 文に続いて，まず，JSD プロセスが，システムの仕様（仕様書記述言語）を引数とし，最終的なシステムの実現方法を表わす図（システム実現図），およびスケジューラプロセスの構造図とサブルーチンの構造図（構造図）の集合を返す関数であると宣言している（注：ここで括弧で括ってあるものはそれぞれの生成物を記述するための言語である）．

前提条件は，入力されたシステムの仕様書に，システムが対象とする世界に関する記述，システムの機能に対する要求，および，システムが作動する環境に関する記述があることである（公理1-1）．

完了条件は，要求仕様にそった目的物を生成すること（公理2-1）であるが，一般には，次に述べる公理 3-1 を満たせば，この条件が満たされると考える．

公理 3-1 は，サブプロセスの関係を表わしている．ここでは，JSD プロセスが，エンティティアクション段階，エンティティ構造段階，初期モデル段階，機能段階，実現化段階の五つのサブプロセスを順次実行すること，および，各プロセス間の入出力の受け渡し関係を定めている．

5.3 エンティティ構造段階の代数的記述

本節では，JSD のエンティティ構造段階の代数的な記述について述べる．

エンティティ構造段階の代数的記述を図２に示す．

テキスト中では，include 文に続いて，まず，エンティティ構造段階を，エンティティリスト（個体の集合），アクションリスト（行動の集合），アクションの順序仕様（仕様書記述言語）を引数とし，エンティティ構造図の集合（個体構造図の集合）を返す関数であると宣言している．

エンティティ構造段階の前提条件と完了条件を以下のように表わす．

(1) 前提条件

前提条件は，入力されたアクションの順序に関する仕様 spec が”アクションの順序に関する仕様か？”という述語関数の値を真にすることである．ここで
”アクションの順序に関する仕様か？”という関数の値の評価は，通常人間が行なう．

(2) 完了条件

必要なすべてのエンティティについて構造図が作成されており，かつ，不必要な構造図が作成されていないことを，述語関数”すべてのエンティティの構造図があるか？”によって表わす（2-1～2-4）．

これを調べるためには，エンティティ構造段階で作成された構造図の集合に含まれる，すべてのエンティティの名前の集合（関数”構造図のエンティティ名の集合”，2-2）と入力されたエンティティリストに含まれる，すべてのエンティティの名前の集合（”関数エンティティリストのエンティティ名の集合”，2-4）とが等しいことを調べればよい．

なお公理 2-2 は，あるエンティティ名の構造図が存在するならば，構造図のエンティティ名の集合の中に，その名前が含まれることを述べている．

この他に，すべてのアクションが対応する構造図内のみに存在することを表す述語関数”すべてのアクションが対応する構造図内にあるか？”，作成されたすべての構造図に表わされている，アクションの発生順序が正しいことを表す述語関数”すべての構造図のアクションの順序が正しいか？”，作成されたすべての構造図が極小構造になっていることを表す述語関数 ”すべての構造図が極小構造か？”が別に定義されている．

6．形式的記述から実行プログラムへの変換例

JSD の開発過程には，機械的に行うことができる事とできない事がある．

実際の開発過程のうち，機械的にできない事が正しく行なわれているかどうかは，人間が調べなければならない．この場合，本報告で述べた定義記述を用いることにより，チェックが容易になる．

機械的にできる事については，定義記述から関数型プログラムを作成し，プログラムを

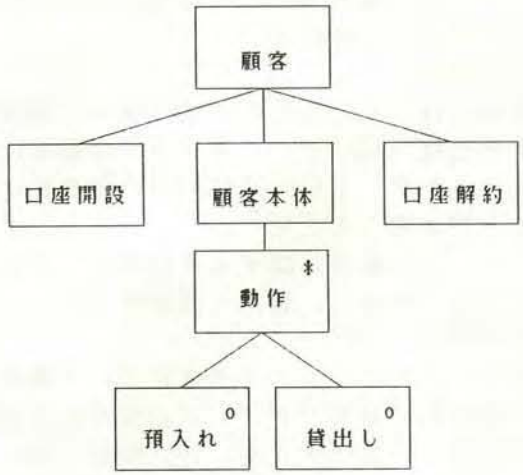
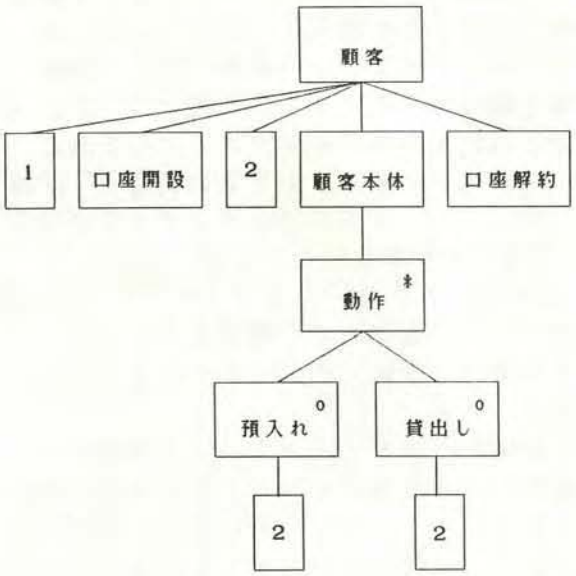


図３　銀行における顧客を表す構造図



1：set-sv（口座開設）
2：set-sv（預入れ，貸出し，口座解約）

図４　set-sv 命令を付加した顧客の構造図

表４　入力順序判定部作成過程の自然語による記述

入力：各モデルプロセスの構造図
　　アクションリスト
出力：set-sv 命令を付加した各モデルプロセスの構造図
　　入力順序判定部
作業：
　　⋮
　　⋮

```
newtxt(s)              ==
    if end-txt(s) then    s
    else if isheadln(line(s),graph(s))
      then newtxt(ins-ln1(nextln(s)))
      else if isactln(line(s),a-list(s))
        then newtxt(ins-ln2(nextln(s)))
        else newtxt(nextln(s));

txt-in(ins-ln2(s))   ==  txt-in(s);
txt-out(ins-ln2(s))  ==
    aput(txt-out(s),out-ln(s),
         aget(txt-in(s),in-ln(s));
in-ln(ins-ln2(s))    ==  in-ln(s) + 1;
out-ln(ins-ln2(s))   ==  out-ln(s) + 1;
graph(ins-ln2(s))    ==  graph(s);
a-list(ins-ln2(s))   ==  a-list(s);
                         ⋮
                         ⋮
```

図５　入力順序判定部作成過程の抽象的順序機械よる記述

```
newtxt(st,gr)   ==
 if end-txt(i_txt(st),i_ln(st)) then st
 else if isheadln(i_txt(st),i_ln(st),gr)
  then newtxt(ins-ln1(nextln(st),gr),gr)
  else if isactln(get_ln(i_txt(st),
                         i_ln(st)),
                  a-list(st))
  then newtxt(ins-ln2(nextln(st),gr),gr)
  else newtxt(nextln(st),gr);

ins_ln2(st,gr)  ==
    [  i_txt(st),
       put_ln(o_txt(st),
              o_ln(st),
              set_first(gr)),
       i_ln(st),
       o_ln(st) + 1,
       a-list(st)  ];
                         ⋮
                         ⋮
```

図６　入力順序判定部作成過程の ASL/Fによる記述

実行して，実際の過程の正しさを調べることができる．

一例として，JSD の開発過程の一つである，入力順序判定部作成過程について述べる．入力順序判定部作成過程は，入力順序判定部をモデルプロセスに付加する，機能ステップの一過程である．

入力順序判定部は，開発しているソフトウェアシステムにおける入力データ系列の順序の正当性を判定する機能プロセスである．

JSD では，実世界における対象を事象の時系列ととらえて，対象を木の根，事象を葉として表わした，正規表現と等価な意味を持つ構造図（例，図３）によって対象をモデル化する．入力順序判定部作成過程は，対象の構造図を入力プロダクトとして，次に起こるべき（入力されるべき）事象の集合を set-sv 命令として構造図に付加（図４）し，set-sv 命令を付加した構造図と入力順序判定部を出力プロダクトとするプロセスである．

入力順序判定部作成過程を定義する際，まず自然語で記述し，次に代数的言語 ASL/1 を用いて抽象的順序機械[15]の形式で記述を行なった後，関数型言語 ASL/F[17] のプログラムに変換した．

表４に示す自然語による記述をもとに，図５に示す抽象的順序機械の形式の記述を行った．ここでは，関数 newtxt によって制御の流れを表わしている．すなわち関数 isactln が真（次に set-sv 命令を挿入する必要がある）ならば，関数 ins-ln2 によって，次に起こるべき事象の集合を求め，状態変数 txt-out に set-sv 命令とともに挿入する．

この抽象的順序機械の記述を図６に示す実行可能な関数型言語 ASL/F のプログラムに変換した．

7．あとがき

本報告では，ジャクソンシステム開発法の定義を試みた．まず，プロセスのモデルに基づいて開発法を分析し，自然語を用いて開発法を記述した後，代数的仕様記述言語 ASL/1 を用いて形式的に記述した．特に，プロセスが何をするのかという事柄と，プロセスをどのようにするのかという事柄を分離したことは，理解しやすい記述を作成する上で有効であったと考える．また，最初にモデルに基づいて自然語で開発法を記述し，その後に形式的に記述するという段階を経ることにより，開発法を正確に記述することができたと考える．

JSD を形式的に記述する作業を通して，JSD の各過程の入出力の関係，および，各過程で行なうべきことを明らかにすることができた．

参考文献

(1) Osterweil, L.:"Software processes are software too", Proc. of 9th ICSE, pp.2-13 (1987).

(2) Kaiser, G.E. and Feiler, P.H.:"An architecture for intelligent assistance in software development", Proc. of 9th ICSE, pp.180-188 (1987).

(3) Barghouti, N.S. and Kaiser, G.E.: "Implementation of a knowledge-based programming environment", Proc. of 21st HICSS, pp.54-63 (1988).

(4) Williams, L.G.:"Software process modeling: A behavioral approach", Proc. of 10th ICSE, pp. 174-186 (1988).

(5) Curtis, B. et al.:"On building software process models under the Lamppost",Proc. of 9th ICSE, pp.96-103 (1987).

(6) Dowson, M.:"Integrated project support with IStar", IEEE Software, Vol. 4, No. 6, pp.6-15 (1987).

(7) Henderson, P.B. and Notkin, D.: "Integrated design and programming environments", IEEE Computer, Vol. 20, No. 11, pp. 12-16 (1987).

(8) Kishida, K. et al.:"SDA: A novel approach to software environment design and construction", Proc. of 10th ICSE, pp. 69-79 (1988).

(9) Parnas, D.L. and Clements, P.C.: "A rational design process: How and why to fake it", IEEE Trans. Software Eng., Vol. SE-12, No. 2, pp. 251-257 (1986).

(10) Riddle, W.E.:"Software designer's associates: A preliminary description", Proc. of 20th HICSS, pp.371-381 (1987).

(11) Jackson, M.A.:"System development", Prentice-Hall (1983).

(12) Cameron, J.R.:"An overview of JSD", IEEE Trans. Software Eng., Vol. SE-12, No. 2, pp. 222-240 (1986).

(13) 松本吉弘：" ソフトウェアに対する要求の形成"，情報処理学会誌，Vol. 28, No. 7, pp. 853-861 (1987).

(14) 松本正雄：" 設計とプログラミングの自動化"，情報処理学会誌，Vol. 28, No. 7, pp. 862-872 (1987).

(15) 嵩，谷口，杉山：" 代数的言語の設計と処理系"，榎本編，ソフトウェア工学ハンドブック，オーム社，pp.93-123 (1986).

(16) 嵩，谷口，杉山，関：" 代数的言語ＡＳＬ／＊ －意味定義を中心に－"，信学論（Ｄ）Vol. J69-D, No. 7, pp.1066-1073 (1986).

(17) 井上，関，谷口，嵩：" 関数型言語ＡＳＬ／Ｆとその最適化コンパイラ"，信学論（Ｄ），Vol. J67-D, No. 4, pp.458-465 (1984).

(18) 飯田元：" ＪＳＤにおける入力順序判定部作成過程の形式的記述"，大阪大学基礎工学部特別研究報告 (1988).

(19) Booch, G.:"Object-oriented development", IEEE Trans. Software Eng., Vol. SE-12, No. 2, pp. 211-221 (1986).

(20) Dahl, O.-J., Dijkstra, E.W. and Hoare, C.A.R.:"Structured programming", Academic Press (1972), 野下，川合，武市（訳）：" 構造化プログラミング"，サイエンス社 (1975).

(21) Jackson, M.A.:"Principles of program design", Academic Press (1975), 鳥居宏次（訳）：" 構造的プログラム設計の原理"，日本コンピュータ協会 (1980).

(22) Myers, G.J.:"Reliable software through composite design", Petrocelli/ Charter (1975), 久保，国友（訳）：" 高信頼性ソフトウェア－複合設計"，近代科学社 (1976).

(23) 野村，井上，鳥居：" システム開発法ＪＳＤの定義付けの試み"，情報処理学会ソフトウェア工学研究会 58-1 (1988).

(24) 野村研仁：" ジャクソンシステム開発法（ＪＳＤ）の定義付けに関する研究"，大阪大学基礎工学部修士学位論文 (1988).