ジャクソンシステム開発法（ＪＳＤ）による情報システム構築について

柴田 宏　，前田 和昭
（慶応義塾大学理工学部）

１．はじめに

ジャクソンシステム開発法(Jackson System Development:JSD)は，M.A.Jackson が1982年に提唱したシステム開発のための手法である．ここでは，まずはじめにＪＳＤの概要を例を示しながら述べる．次に，我々がプロトタイプとして作成しているＪＳＤによる開発を支援するためのツール(JAckson system development Support Tool:JAST)を紹介し，最後に，ＪＡＳＴの支援ツールとしての今後の在り方について，ＪＳＤの特徴を踏まえながら検討する．

２．ＪＳＤの概要

２．１　ＪＳＤの基本概念

M.A.Jackson は，1970年代に構造的プログラム設計法(Jackson Structured Programming:JSP)[1,2]を開発している．ＪＳＰはその名の通り，プログラム設計のための手法である．それに対し，ここで述べるＪＳＤ[1,3]は，情報システム開発過程全般にわたってその開発を導いていくために提案された方法で，その一部にＪＳＰの手法を用いており，また記法も共通のものが多い．

JacksonがＪＳＤを提唱するにあたって強調した点は次の２点である．

（ｉ）モデル(model)の重視
（ii）仕様(specification)と実現(implementation)の分離

まず(ｉ)について述べる．ＪＳＤにおいてモデルとは，これから開発しようとしているシステムにかかわりのある現実世界(real world)のモデルのことを指す．ＪＳＤではシステムの機能(function)の記述を始める前に，必ずモデルの記述を行う．そのモデルはシステムに,その基礎をなす部分として組み込まれる．モデルは現実世界から情報を受け取り，絶えず現実世界をシミュレートする．システムにおける機能は，そのモデルから情報を受け取ることにより，そのシステムとしての機能を発揮する．すなわち，ＪＳＤではシステム内においてモデルと機能を分離して考える．これを図に表すと，図１のようになる．

システム開発においてこのようにモデルを重視する第一の理由は，モデルが機能の変更に対して安定しているからである．例えば，図１において機能１に変更があった場合を考える．その時，モデル，機能２は機能１の変更に影響されない．また，新しく別の機能が加わった場合も，モデルや既存の機能の変更は最小限にとどめることができる．そこで，このような機能の変更に対して安定しているモデルを重視し，機能とは分離して考えるわけである．

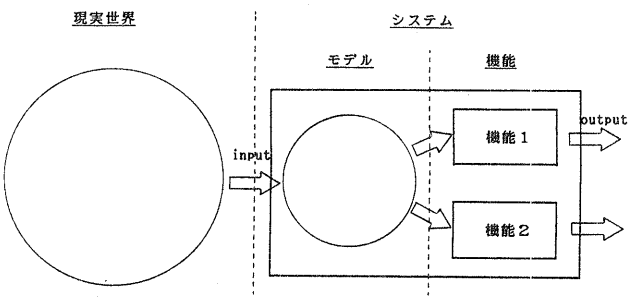


図１．ＪＳＤにおけるシステム

また第二の理由として，ユーザがそのシステムに対する要求（すなわち機能）を開発の当初に明言するのが難しいということが挙げられる．ユーザの要求とは本来曖昧なものであり，システムを開発していく過程において次第にはっ

きりして来るものである，と言える．そのため，いわゆる要求分析を行う前段階として，現実世界に対するユーザの見方，すなわちモデルを記述することを行う．その時にユーザが参加していれば，その過程においてユーザの要求も明確化してくるであろう，というのがＪＳＤの考え方である．

このように，ＪＳＤではモデルを重視しているわけであるが，前述のようにモデルとは現実世界のモデルのことを指しているのであるから，現実世界を何らかの形で抽象化しなければならない．そこでＪＳＤでは現実世界に存在するエンティティ(対象)に注目し，それがかかわるアクション(事象)を時間的順序をもって記述することにより抽象化を図っている．

次に(ii)について述べる．ＪＳＤによる開発は次の３つの局面(phase)から成る(図２)．

- モデル
- 機能
- 実現(implementation)

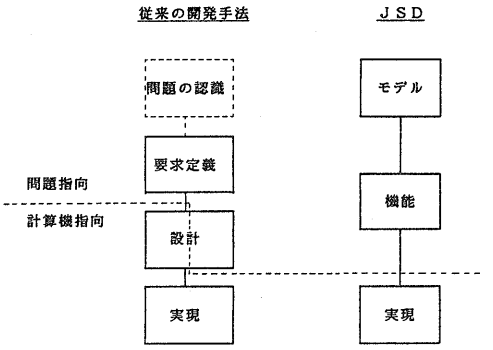


図２．従来の開発手法との比較

(i)で述べたように，まずモデルに関する仕様が作られ，次に機能に関する仕様が作られる．この２つの局面は，仕様作成のための局面であり，次の実現の局面とは大きく分けられる．ここまでで作成される仕様は，名目上実行可能な形式を持ったものである．すなわちオペレーショナルな仕様である[5]．しかし，それはいま対象としている問題に合わせた構造をしており，計算機上で効率良く実行できるかどうかは無視している．実現の局面はそのような仕様を計算機上で効率良く実行できるように変換する部分である．

このように仕様と実現を分けた第一の理由は，ＪＳＤが問題指向的な部分と計算機指向的な部分との分離を目指したからである．モデルと機能という２つの局面は，要するに"ユーザが何を要求しているのか"を決定する部分である．従って，少なくともこの２つの局面までは，ユーザの見方，すなわち問題指向的な見方で仕様を書かなければならない，というのが Jackson の主張である．計算機に対する考慮は実現の局面において考える．

仕様と実現を分離した理由の第二として，従来のシステム開発手法の多くにみられた，開発各段階（要求定義→システム設計→実現）間におけるギャップの問題を挙げることができる．例えば要求定義においては，システムが何をすべきか(what)という要求を定義するのに対して，システム設計においては，その what をどのようにして達成すべきか(how)ということを考えるわけであるが，その how が what を本当に満たしているかどうかの検証は難しい．また，どこまでが what でどこからが how か，という区別も曖昧である．これはシステム設計／実現 間においても同様である．ＪＳＤではこのような問題の解決策として，what／how の分離の代りに，仕様／実現の分離を考え，仕様は始めから how を記述することを提案している．すなわちオペレーショナルな仕様を書くことを提案しているわけである．もっとも，この解決策に問題がないわけではない．これについては後述する．

２．２　ＪＳＤの開発ステップ

この節ではＪＳＤの開発ステップについて例を示ながら説明する．ここでは，外国図書を購入，顧客の注文に応じて販売している会社において，事務の効率化，顧客に対するサービス向上等のため，顧客関係の情報を提供するようなシステムの開発を例に挙げる．この例は我々が実際に分析・記述したものであるが，簡単のためその一部についてのみ示すことにする．

ＪＳＤの開発ステップを図３に示す．この図は，前述の Jackson の主張を反映した構造になっている．開発はこの木の葉の部分を左から右へと進む．以下に各ステップの内容について述べる．

（ｉ）エンティティ アクションステップ (entity action step)

このステップでは，エンティティ(対象)と，そのエンティティにかかわるアクション(事象)とをリストアップする．エンティティとは，これから開発するシステムの外の現実世界に存在する"もの"で，それは具体的なものであっても，概念的なものであってもよい．

このステップの仕事はこのエンティティとアクションのリストアップだけであるので，一見簡単である．しかし，このステップで，システムがかかわる現実世界を把握し，それはまた，システムの捉えている範囲を決めることになるので，そのリストアップは慎重に行わなければならない．

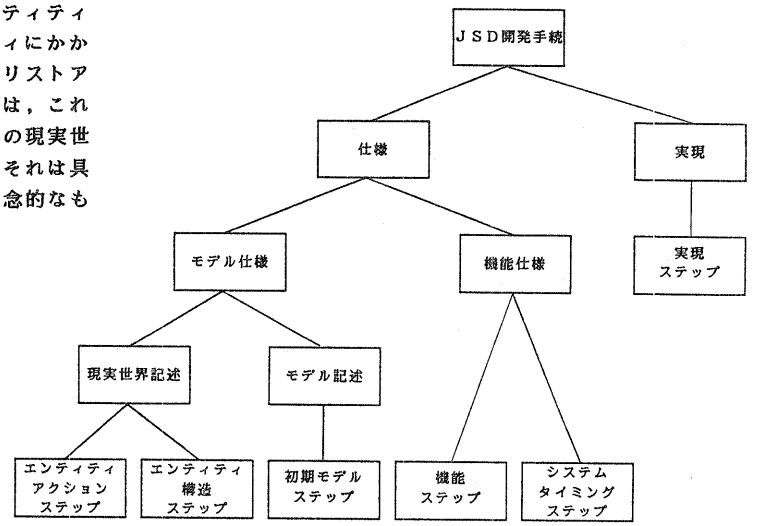


図３．ＪＳＤの開発ステップ

前述の例について，このステップにより作成されるドキュメントを図４に示す．注意すべきことは，ここでは紙面の都合上示さなかったが，エンティティ・アクションとして採用されなかった対象・事象についても，それがどのような理由で採用されなかったかという記録をドキュメントとして作成しなければならない，ということである．これから先の開発過程において，なぜその"もの"をエンティティとして採用しなかったのか，という問題は必ず生じる．また，システムの捉えている範囲について，ユーザとのもめごとも起こりそうである．そのようなときに，これらのドキュメントを残しておくと，有用である．すなわちドキュメントとは，開発の過程を記録するものであり，開発の結果を記録するものではない．これは以後の開発ステップにおいても同様である．

エンティティ	アクション	アクションの説明
顧客	図書注文	顧客が店頭・電話・郵便等で図書を注文する
	注文取消	顧客が注文を取り消す
	入金	顧客が入金をする
販売課	発注	顧客の注文に対し在庫がないとき発注する
	知らせ	発注した図書が入庫したときそれを顧客に知らせる
	納品	注文の図書を納品する
	請求	顧客にその注文に対する請求をする
	領収	顧客にその注文に対する領収書をわたす
注文	(顧客・販売課のアクションと共通)	

図４．エンティティ・アクションのリスト

（ⅱ）エンティティ構造ステップ(entity structure step)

このステップでは，前ステップで決定した各エンティティについて，そのアクションの時間的順序を表した構造図(structure diagram)を作成する．

図５は先の例において，エンティティ"注文"について記述された構造図である．アクションは木の葉で表されており，左から右へと進む．図中において∘印は選択(selection)を表す．また －印は何も行われないことを示す(null action)．

(iii) 初期モデルステップ (initial model step)

前ステップまでで現実世界の抽象的記述が完成した．その記述をもとに，モデルの仕様を完成させ，それを現実世界と結合させるのがこのステップである．それによりモデルの状態は現実世界に追随して変化(シミュレート)する．モデルの仕様は各エンティティについて逐次プロセス(sequential process)の形式で表される．これを構造テキスト(structure text)と言い，前ステップで記述された構造図から作られる．このステップで作成するドキュメントは，現実世界とモデルとの結合状態を表したＳＳＤ(System Specification Diagram)と，各エンティティに関する構造テキストである．

図６，図７はそれぞれ先の例におけるＳＳＤと，エンティティ"注文"に関する構造テキストである．図６において ▭ は，現実世界においてはエンティティ，モデルにおいては逐次プロセスを表す．→○→ はデータストリーム結合(data stream connection)を表し，矢印方向がデータの流れる方向である．矢印上における ⊣⊢ は複数のデータストリーム結合があることを表す．この例では，例えば，顧客-1プロセスと注文-1プロセスとは 1:n の関係があることを表している．また，図７の構造テキストは，図５の構造図とその構造が一致している．

(iv) 機能ステップ(function step)

このステップに入って初めてシステムの機能を記述する．機能は現実世界をシミュレートしているモデルからその情報を受け取ることにより，システムとしての機能を果たす．このステップで作成されるドキュメントは，機能の記述を加えたＳＳＤと，その機能の内容を記述した構造テキストである．この構造テキストも逐次プロセスの形式で記述される．構造テキスト作成にはＪＳＰを用いる．

図８，図９はそれぞれ先の例に関するＳＳＤ，構造テキストである．ここで作成された機能は，毎月１回，顧客への請求金額をレポートとして出

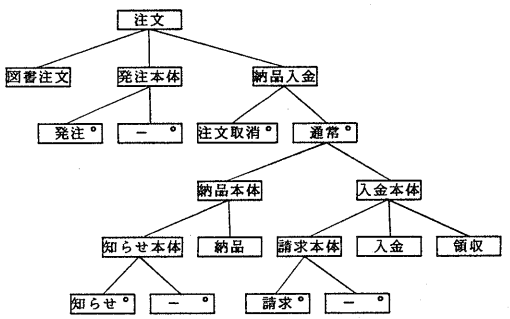


図５．エンティティ"注文"の構造図

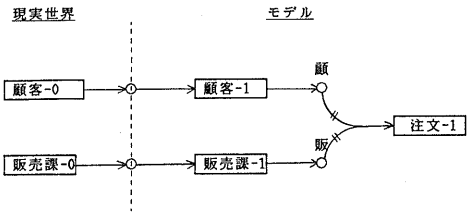


図６．初期モデルステップにおけるＳＳＤ

```
注文-1 seq                          /* seq は sequence を表す  */
    read 顧&販;
    図書注文; read 顧&販;
    発注本体 sel (発注)             /*  sel は selection を表す  */
        発注; read 顧&販;
    発注本体 alt (otherwise)        /*  alt は alternation を表す  */
    発注本体 end
    納品入金 sel (注文取消)
        注文取消;
    納品入金 alt (otherwise)
        納品本体 seq
            知らせ本体 sel (知らせ)
                知らせ; read 顧&販;
            知らせ本体 alt (otherwise)
            知らせ本体 end
            納品; read 顧&販;
        納品本体 end
        入金本体 seq
            請求本体 sel (請求)
                請求; read 顧&販;
            請求本体 alt (otherwise)
            請求本体 end
            入金; read 顧&販;
            領収;
        入金本体 end
    納品入金 end
注文-1 end
```

図７．エンティティ"注文"の構造テキスト

力するものである．図８における →◇→ は状態ベクトル結合(state vector connection)を表す．データストリーム結合との違いは，イニシアチブがデータの受け手側にあることである．ＪＳＰにより構造テキストを作成する過程は，ここでは省略する．

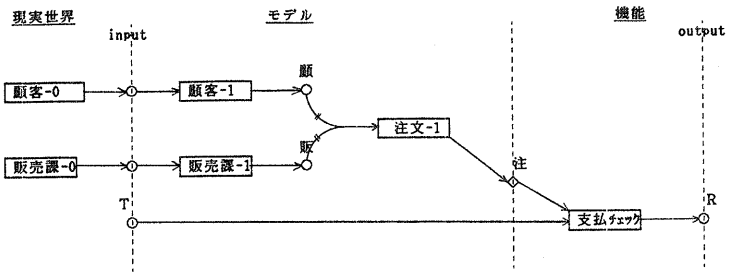


図８．機能ステップにおけるＳＳＤ

（ｖ）システムタイミングステップ (system timing step)

前ステップまでの仕様は，タイミングに関する要求（応答時間，オンライン処理かバッチ処理か等）が明示されていなかった．このステップでは，それをコメントとして記述する．このコメントは，前ステップまでの仕様を，次の実現ステップにおいて変換する際の指針となるものである．

```
支払チェック seq
    read T;
    支払チェック本体 itr                /*  itr は iteration(繰り返し)を表す  */
        write HEADER to R;
        getsv 注;
        期間 itr while (not EOF)
            stored顧客No = 顧客No;
            請求合計額 = 0;
            顧客 seq
                write 顧客No to R;
                注文本体 itr while (顧客No = stored顧客No)
                    未納 sel ( 納品 <= tp < 入金 )
                        請求合計額 = 請求合計額 + 請求額;
                        write 注文No,請求額 to R;
                    未納 end
                    getsv 注;
                注文本体 end
                write 請求合計額 to R;
            顧客 end
        期間 end
        write TRAILER to R;
    支払チェック本体 end
支払チェック end
```

図９．機能＂支払チェック＂の構造テキスト

（ｖｉ）実現ステップ (implementation step)

前ステップまでで完成した仕様は，モデル，機能とも，ネットワーク化された逐次プロセスの形式で記述され，実行可能な形式を持っている．しかしそれは計算機上での実行効率を無視したものである．そのような仕様を計算機上で効率的に動くプログラムに変換するのが，このステップである．従って従来のシステム開発手法(図２)における実現段階とは性質を異にする．

このステップでの主な仕事は以下に示すものである．

- スケジューリング(scheduling)
- 構造テキストから状態ベクトル(局所変数)を分離(state vector separation)

ＪＳＤでは，仕様上においては多くのプロセスが並行に動くことを仮定している．先の例では，"顧客-1"プロセス，"注文-1"プロセス等が現実世界における顧客，注文の数だけ存在し，それが並行に動いている．その並行に動いているプロセスを，そのままの形で単一のプロセッサしか持たない計算機上で実行させるのは不可能であり，また，それが実現できても，共通のテキストを持つ多数のプロセス（例えば，顧客-1プロセスは全て共通のテキストを持つ)を並行に動かすのは，膨大なメモリを必要とする．そのため実現ステップでは，各プロセスが交互に動くように，各プロセスを変換して制御を行い（スケジューリング)，また，共通のテキストを持つプロセスから状態ベクトルを分離して，共通のテキストはひとつだけ持てばすむようにさせる(状態ベクトルの分離)．

以上がこのステップでの主な仕事であるが，他にもこのステップでは様々な仕事をしなければならない．それは，システムが実現される環境によって異なり，先のシステムタイミングステップで示された要求によっても異なる．その詳細はここでは省略する．

３．ＪＳＤ開発支援ツール

前章では，ＪＳＤの基本概念と開発ステップの概要について述べた．この章では，我々が現在設計・開発を進めている，ＪＳＤによる開発を支援するためのツール：ＪＡＳＴについて述べる．

３．１　ＪＡＳＴによる開発支援

前章で示したように，ＪＳＤでは各開発ステップで作成されるドキュメントが明確に定められている．各ドキュメントはその開発ステップで行われた決定事項の記録であり，後の開発過程において参照される．従ってドキュメントは参照が容易なように整理されなければならない．ところが２．２節(ｉ)でも述べたように，ドキュメントは各開発ステップの結果を記録したものではなく，開発の過程を記録したものでなければならないから，ドキュメントのバージョンを多数管理することになる．また，構造図と構造テキストの関係(２．２節(ⅲ)参照)のように，ドキュメント間の一貫性も保たねばならない．これらのことを人手だけで行うことはかなり困難であり，システム開発以外のところで余分な労力が必要となる．これを計算機で支援することで，システム開発者の労力を軽減し，システム開発本来の仕事に専念できるようにすることが，ＪＡＳＴの目標とするところである．

ＪＡＳＴは以下に示す指針をもとに設計を行った．

（ｉ）ＳＳＤを中心にした各ユーティリティの統合
（ⅱ）ドキュメント間の一貫性の保持
（ⅲ）ドキュメントのバージョン管理

まず(ｉ)について述べる．ここでユーティリティとは，ＪＳＤの各ドキュメントを作成，更新させるためのエディタのことである．ＪＳＤではエンティティに注目して現実世界を捉えており，ＳＳＤではそのエンティティを  で表す．また，初期モデルステップ，機能ステップで作られるプロセスも同様に，  で表現され，エンティティ，プロセス間はネットワークで結ばれる．従って，ＳＳＤを見ればシステム全体を捉えることができる．そこでＪＡＳＴでは，ＳＳＤを中心として各ユーティリティを統合するように設計している．すなわち，ＳＳＤ作成ユーティリティから，ＪＡＳＴの他の全てのユーティリティを起動させ，ＪＳＤの各ステップにおける開発を支援できるようにした．

(ⅱ)，(ⅲ)については，先にも述べたように，開発過程におけるドキュメント管理の軽減化を目指したものである．

３．２　ＪＡＳＴを構成する各ユーティリティ

ＪＡＳＴは，次の４つのユーティリティで構成されている．

（ｉ）ＳＳＤ作成ユーティリティ

ＳＳＤ作成ユーティリティは，ＳＳＤを対話的に作成するためのユーティリティである．ＪＡＳＴでは，このユーティリティが主となっており，他の３つのユーティリティはここから起動される．

（ⅱ）関連図作成ユーティリティ

関連図とは，エンティティとアクションの関係をグラフで表現したものである．関連図作成ユーティリティは，この関連図を対話的に作成するために使用する．

（ⅲ）構造図作成ユーティリティ

構造図作成ユーティリティは，構造図を対話的に作成するためのユーティリティである．このユーティ

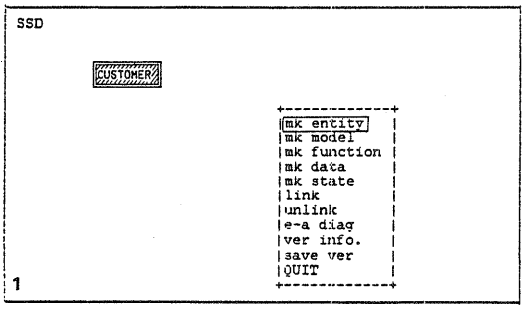
リティの終了時には，関連図との一貫性をチェックする．すなわち，構造図の葉にあたる部分が，関連図で定義したアクションか，もしくはヌル(null action)のどちらかであることをチェックする．

（iv）構造テキスト作成ユーティリティ

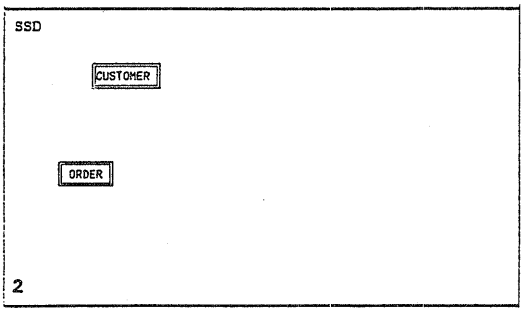
構造テキスト作成ユーティリティは，構造テキストを作成するためのユーティリティである．このユーティリティは，(ii)で作成された構造図から構造テキストの制御構造を自動的に作成する．従って，(iii)の構造図作成ユーティリティで行われた修正は，必ず構造テキストに反映される．逆に，構造図に修正を必要とする構造テキストの修正は，このユーティリティでは禁止している．また，編集対象が構造を持っているので，このユーティリティには構造エディタ特有の機能（ホロフラスティング，ズーミング）を持たせることができる．

３．３　ＪＡＳＴの使用例

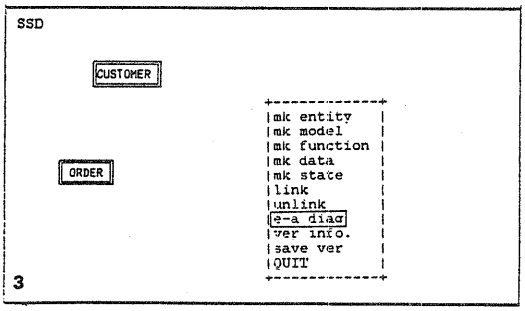
ここではＪＡＳＴの使用例をしめす．各ページの左側の図は，ＳＳＤ作成ユーティリティの画面，右側の図は，その左側のＳＳＤ作成ユーティリティから起動されたその他のユーティリティの画面である．また，ＳＳＤ作成ユーティリティ画面上にある ▭ はエンティティ， ▭ はモデル， ▭ は機能を表す．

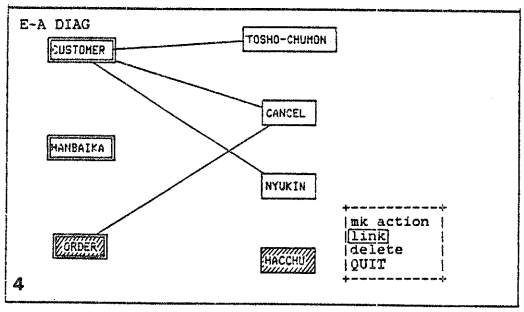


1→2
ＳＳＤ作成画面上でポップアップメニューを出して，"mk entity"(エンティティの作成)を選択・実行する．



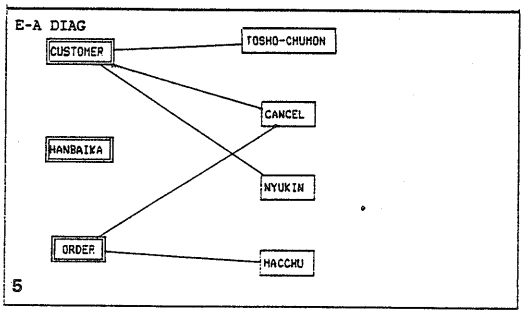
3→4
ＳＳＤ作成画面上で，"e-a diag"(関係図)を選択・実行する．

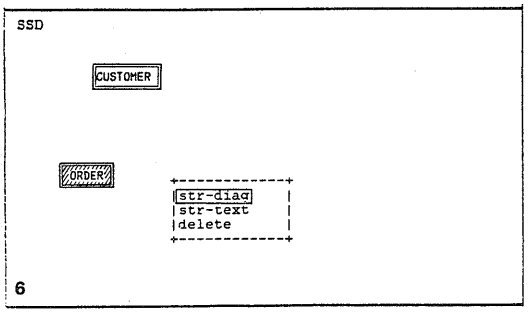


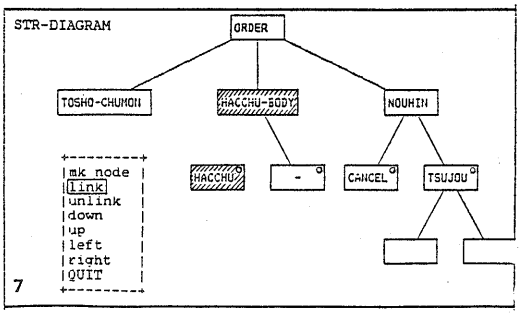


4→5

"link"を選択・実行し，"ORDER"と"HACCHU"を選択する．





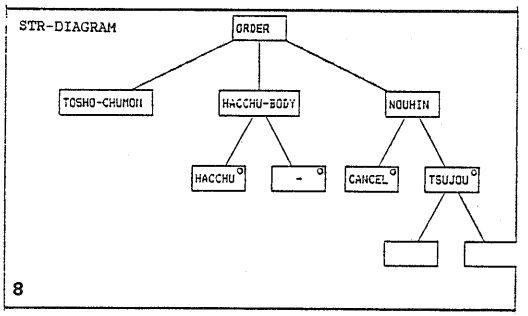


6→7

"ORDER"を選択して，ポップアップメニューを出し，"str diag"(構造図)を選択・実行する．

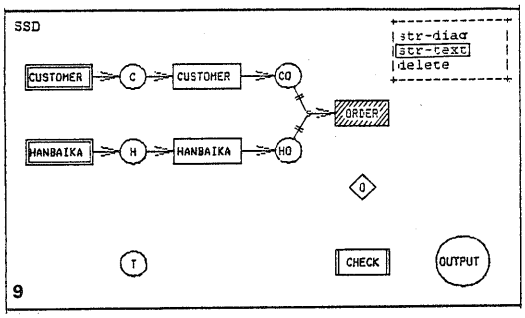
7→8

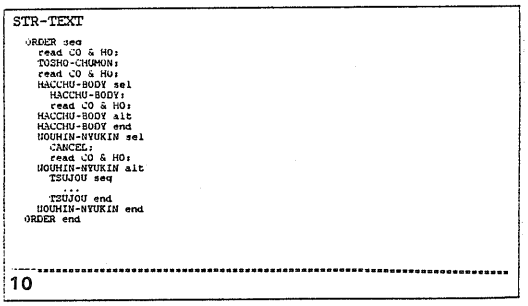
"link"を選択・実行し，"HACCHU-BODY"と"HACCHU"を選択する．

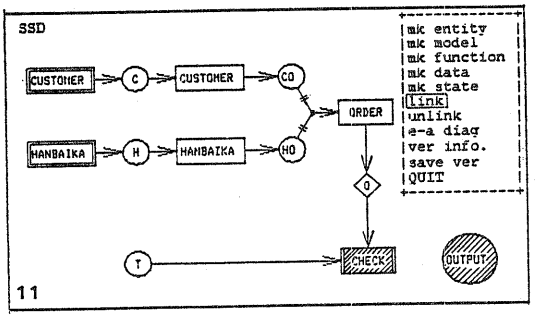


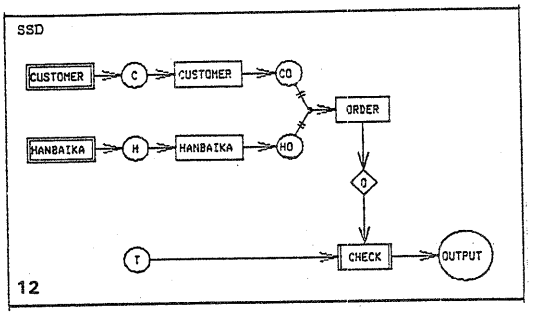
9→10

"ORDER"を選択し，"str text"(構造テキスト)を選択・実行する．





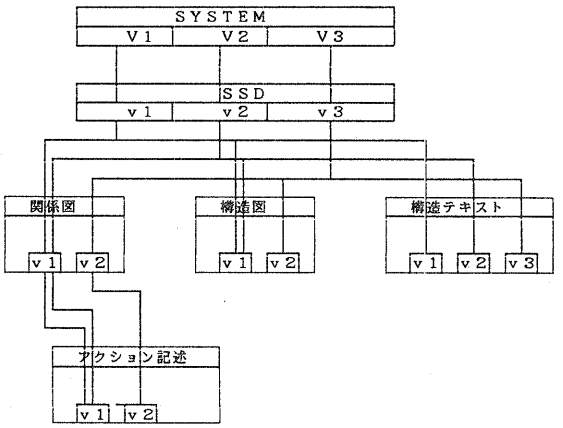




１１→１２
"link"を選択・実行して，"CHECK"と"OUTPUT"を選択する．

３．４　ＪＡＳＴのバージョン管理

これまで述べてきたＪＡＳＴは，各ドキュメントの変更の履歴を保存している．各ドキュメント間の関係と各々のバージョンを管理するために，システムの構成を右図のようなグラフで表現する．



３．５　検討

（ｉ）現ツールに関して

ここでは，３.１-３.４節で紹介したＪＡＳＴについて検討する．

我々がＪＡＳＴで目指したことは，ドキュメント管理の軽減化である．バージョン管理，ドキュメント間の一貫性の保持といった点については，現ツールの段階で満足いくものであると考えられる．また，ＳＳＤを中心とした各ユーティリティの統合もシステム全体が見やすくなるという点において適切である．しかし，現ツールをより使いやすいものにするためには，次のような点を改良していかなければならない．

現在動いているツールは，簡易グラフィック機能を持つターミナル上で実現されているが，それを高機能ワークステーションに移植することにより，環境の改善を図る．それと同時にマルチウィンドゥ，日本語への対応も考えていかなければならない．

ＪＡＳＴは現在，部分的に実現されているが，今後，ツール全体の完成を目指し，その後に本格的な評価をしていくつもりである．

（ii）今後の支援ツールの在り方について

（i）で述べた範囲が現在考えているツールの全体像であるが，今後，より広い範囲を支援するツールを考えていくためには，ＪＳＤの持つ特徴について詳しく検討しなければならない．そこで，ここではＪＳＤの特徴を考察しながら，支援ツールの今後の在り方について考えてみる．

前章で述べたように，ＪＳＤは what と how の分離に代わるものとして，仕様と実現の分離を目指した．それにより仕様は，システムの機能を考える段階までは，問題指向，すなわち問題領域に合わせた形式で記述することになった．しかしながら，このことに関して考えなければならない点が２つある．

第一に，ＪＳＤの仕様がシステム内の動きを詳述したオペレーショナルな仕様であるので，システム全体の外から見た動きがわかりづらい点が挙げられる．ＪＳＤでは，システム全体の動きがわかるようなドキュメントとしてどのようなものを作ったらよいか，ということにはふれていない．それはＪＳＤの対象としている問題領域が広範囲にわたるためである．問題領域ごとに，それに適したドキュメントは異なる．例えば，会話型の情報システムでは，会話を行なう画面のイメージに関するドキュメントがあれば，システムのおおよその動きはわかるが，エレベータ管理システムのような埋込型のシステムでは，そのようなものは適さない．それよりもシステム全体の動きを知るには，その仕様をもとに何らかのシミュレーションを行うのが適している．

従って，ＪＳＤによる開発をこの点において支援するためには，問題領域ごとに別々のツールにするのが適当であると思われる．現に，埋込型システムに問題領域を絞って開発を支援するツールが既に考えられつつある[4]．

第二の点は，実現ステップに関することである．ＪＳＤにおける実現ステップは，前ステップまでの仕様を効率よいコードに変換する部分である．ここは，人手で行うには，かなり複雑な作業を必要とする．そこでツールによる支援が考えられるが，現段階においてそれを実現するには，多くの問題を解決していかなければならない．この点は，今後検討を要する課題である．

４．おわりに

本報告では，はじめにＪＳＤの概要について述べ，次にそのＪＳＤに基づいて我々が開発しているシステム開発支援ツールを紹介した．プロトタイプとして設計したこの開発支援ツールを完成させ，今後それに対する評価を加えていくつもりである．また，４．４節(ii)でも述べた支援ツールとしての今後の在り方について考察を深め，同時に，それを通して，方法論としてのＪＳＤに関しても再検討をしていきたいと考えている．

参考文献

[1] Cameron,J.R.: JSP & JSD:The Jackson Approach to Software Development , IEEE Computer Society,Long Beach,California(1983)

[2] Jackson,M.A.: Principles of Program Design , Academic Press,London(1975).

[3] Jackson,M.A.: System Development , Prentice Hall,Englewood Cliffs,New Jersey(1982).

[4] Potts,C.,et al.: Discrete Event Simulation as a Means of Validating JSD Design Specifications , Proceedings of the 8th International Coference on Software Engineering,pp.119-125(1985).

[5] Zave,P.: The Operational Versus the Conventional Approach to Software Development , Communications of the ACM,vol.27,no.2,pp.104-118(1984).