解　説

ジャクソンシステム開発法†

大　野　侑　郎††

1. 位置づけ

ジャクソンシステム開発法は，M. A. Jackson のジャクソン構造的プログラミングの拡張である[1),2),4)]．

前者は後者を含み，どちらもジャクソン法と呼ばれている．Jackson は，「古典的ライフサイクル論」を認めないから，前者が後者のライフサイクル上流，とくに「設計」部にあたるという誤解を何度も拒んできた．そもそも「設計」というあいまいなタームを彼は'80年代には用いていない．

ジャクソンシステム開発法が扱う対象は，

（1） モデル (model)

（2） 機能 (function)

（3） 実現 (implementation)

の3つの局面であり，実世界 (real world) は(1)→(2)→(3)のように変換されて，システム開発が行われる．

逆に，扱わない対象は，実現性分析（フィージビリティスタディ），システム開発の正当化や優先順位づけ，プロジェクト管理などである．また，方法論 (methodology) にも言及しない．

ジャクソン法の根拠は，逐次プロセス間通信/相互通信逐次プロセス (CSP, Communicating Sequential Process) の理論である[3)]．1978年に C. A. R. Hoare が CSP を提唱し，プログラミングでは入力，出力，併行性が基本的概念 (Primitive) であることを強調した．別に，有用な概念としてモニタなどもあるが，これらは CSP により明示的に定式化できることを示した．ジャクソン法はこのような視方を，仕様を具体化する以前のモデリングの過程にまで拡張したシステム開発法である．

2. 概　要

図-1 のように実世界は，システム化すべき対象世界である．モデルは，後述する逐次プロセス間通信の視点から対象と事象によって実世界を抽象化したものである．実世界とモデルには情報の通信があり，どちら（の仕様）もインタプリティブに実行可能 (executable, operational) である．システムは，モデルに対する機能の追加であり，システム化することによって始めて必要となる，例えば診断情報出力機能などを追加する．システムも実行可能である．

実現は，計算機資源を考慮したシステムの最適変換の結果である．モデル，システム，実現の間にも情報の通信がある．そしてどれも CSP で記述され，実行可能である．

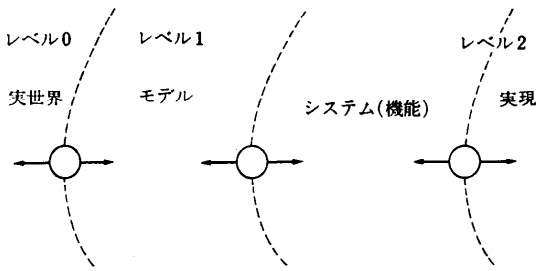


図-1　ジャクソンシステム開発法のスコープ

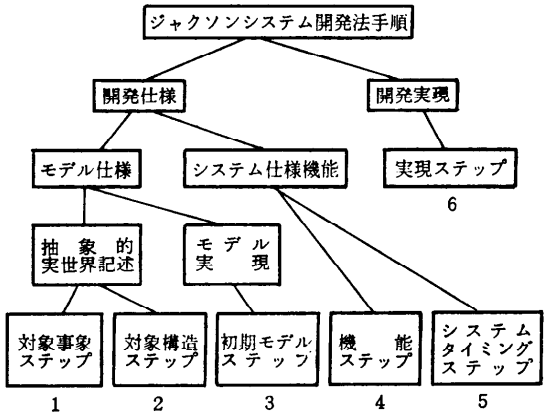


図-2　ジャクソンシステム開発法ステップ

† Jackson System Development by Toshiro OHNO (Japan Business Automation Inc.).

†† 日本ビジネスオートメーション(株)

ジャクソンシステム開発法のステップは，図-2 のとおりである．図の樹構造は，まず根の次の左側の分枝が先，右側の分枝が後という前後関係にあり，そして次の次の分枝についても同様…のように読む．

ステップ 1〜3 は，システム化すべき実世界の対象の CSP に基づくモデル化に係わる．ステップ 4，5 は機能とシステム出力とタイミングに係わり，従来の「設計」概念に近い．ステップ 6 はプログラム実行の前段階であり，対象コンピュータを想定して実行可能なプログラムを作成する．これをもとに，ジャクソン法の処理系 (JSP-Cobol など*) によって，コードが自動作成される．

2.1 対象事象ステップ (entity action step)

対象事象ステップでは，作成すべきモデルの境界を定め，その中で着目する実世界の対象 (entity) と事象 (action) をリストアップする．このステップの出力は次の 3 つの情報である．

(1) 対象リスト (entity list)

(2) 事象リスト (action list)

(3) 事象記述 (action discription)

対象は，実世界に実在する個 (individual) で唯一の名辞をもつ．そして時系列上で 1 つの事象を伴う．

事象は，実世界で生起するものに限り，個的(atomic) で部分事象には分割できず，また合成事象も許さない．事象は，対象の一時点 (instantaneous) におけるものに限り，状態は許さない．合成事象や状態は，後のステップでモデルの個別プロセスの合成や状態として定式化される．

対象の選択には，次の 2 つの制限を考慮する．

(a) 必ず事象を時系列で伴う．

(b) 対象のライフタイムのなかでタイプは変らない．

もし対象のタイプや役割り (role) が変るときは，モデルの外へ出してしまうか，異なる役割りを含む上位の対象を設定する．集合的 (collective) な対象や汎用的 (generic) な対象を扱うときには注意が必要である．

事象記述は，形式的でない自然言語によって事象に定義と属性を与えるものである．定義と属性は最小なものとし，属性は必ず変数で表示する．共通事象や例外的事象はこれを明示しておく．

2.2 対象構造ステップ (entity structure step)

対象の構造を事象の時系列の視点から樹構造で表示する．事象は，逐次的 (sequential) であるか選択的 (selective) であるか反復的 (iterative) であるかのどれかとする．実世界を抽象化した必要十分な構造を設定すればよく，いわゆる「機能」たとえば診断情報の出力などはこれを含めないで，後の機能ステップで扱う．

樹構造で表示しにくい例がある．1 つは，ある対象から別の対象が誕生するような場合で，例えば銀行口座で「名寄せ」を要するようなケース．もう 1 つは対象が複数の役割りをもち単一の樹構造で表示できないケース．またもう 1 つは，選択や反復に必要な条件式が，選択肢や反復事象そのものを実行しないと評価できないケース．これらのケースについて，ジャクソン法では複数のプロセスに分解する，後戻り (backtracking) を行うなどの定型化された解決手法を提供している．

2.3 初期モデルステップ (initial model step)

実世界とモデルを峻別し，しかもモデルが実世界をよく反映するように，実世界とモデルのプロセスの結合に次の 2 つを考える．

(1) データの流れ結合

図-3 データの流れ結合

(2) 状態ベクトル結合

図-4 状態ベクトル結合

データの流れ結合では，プロセス P がメッセージないしレコードの順序集合からなる逐次データの流れを書き出し，プロセス Q がそれを読み込む．「書出し」の順序が「読込み」で保存されるプロセス間通信である．

状態ベクトル結合では，P の状態 (P がテキストのどこまで進んだかを示す内部変数) を Q が直接調べる．例えば，コンパイラのシンボルテーブルなどがこの例である．

(1), (2) の違いは，イニシアティブを P と Q のどちらがとるかに係わっている．単純化した例を示せば図-5 のようである．

眠りの深い人なら，目覚時計のアラームで眼を覚ま

---

* これらの処理系の販売は，日本ビジネスオートメーション(株)で行われている．

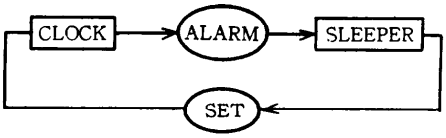


図-5 イニシアティブ

す（データの流れ結合）．眠りの浅い人は，短い寝覚めごとに時計の針をチェックする（状態ベクトル結合）．

このステップの出力は，システム仕様図とそれに基づく構造テキストである．

2.4 機能ステップ (function step)

モデルにおいては，実世界の事象の発生 (event) を報らせるメッセージだけがモデルへの入力であり，実世界の構造がモデルにシミュレートされていた．

機能ステップでは，モデルに機能を追加してシステム仕様図がつくられる．機能はシステムの出力により特徴づけられる．まずモデルとシステムの間のプロセス間でやはりデータの流れ結合と状態ベクトル結合が選択される．また，システムの出力機能を明示し，入力機能も再構成する．さらに，作成したシステムがモデルの変更を伴うときシステムからモデルへのフィードバック機能を追加する．

このとき，機能は必ず逐次プロセスに基づいて指定され，手続き (procedure) の概念は採らない．機能の追加にあたって，複数入力の問題，排他制御，後戻りなどが問題となることがある．

このステップの出力は，システム仕様に必要な機能を追加した拡張システム仕様図である．

2.5 システムタイミングステップ (system timing step)

システム仕様ではタイミングを明示的に導入しなかった．このステップで始めて，タイミングあるいは同期プロセスが扱われる．

タイミングの遅れには，実世界からモデルに至る情報の時間差，一括処理等による時間差，機能プロセスの同期のための時間差などがある．そこでこのステップでは，例外レポートの出力，問合せレポートの出力，業務時間の制約，端末の応答速度などをインフォーマルに記述する．場合によっては，時間差の処理のために同期プロセスを明示的に導入することもある．

2.6 実現ステップ (implementation step)

システム仕様を計算機環境で実行させるために，システム仕様のスケジューリングと実行の効率化を狙った変換を行う．このステップの出力として，システム実現図と実行可能プログラムがある．

システム実現図では次が明示されなければならない．

（1） プロセッサの数
（2） プロセッサへのプロセス配分法
（3） (2)のスケジューリング法
（4） 転置 (inversion) によるスケジューリング
（5） バッファによるスケジューリング
（6） プロセスの分割
（7） 状態ベクトルの分割
（8） データベース

ここまでのシステム仕様は，ある意味で実行可能になっていた．つまり，計算機資源が充分大きくかつ CSP が動けば，それは実行可能であった．初期モデルやシステムについても同様である．

現実の計算機資源に合わせるには実現の工夫が必要であり，しかも正しい変換による正しい実現を図りたい．そうすれば，システム仕様が正しければ実現されたシステムも正しいことになる．

ここに若干の問題がある．1つはモデルの併行性であり，もう1つは同種のプロセスが多数あることである．前者の解決にはデータの流れ結合のとき，疑似コルーチン化を図る．例えば前処理系 JSP-Cobol はこの機能をもっている．後者の解決については，同種のプロセスは同じテキストをもつから，プロセス数と同数の局所変換をもたせて再入可能 (reentrant) コードにする．

(4)の転置は，プロセスを中断-復帰機構をもつ可変状態手続き (procedure) に変換するものである．このように手続き化されたプログラムが呼ばれると，呼ばれた手続きはその最初から実行を始め最初の中断点 (send または receive) まで実行を続け，そこで呼んだプログラムに制御が戻る．次の呼びだしのさいは，手続きの入口のスイッチによって始めの中断点の次に制御が戻り次の中断点まで実行し，また呼んだプログラムに制御を戻す．これが疑似コルーチンの動作である．図-6(a)の例に対し，プロセス Q を P の手続き（疑似コルーチン）化するのを図-6(b)のように示す．

一般にプロセスは同じ類の具体プロセスの集りである．それぞれのプロセスは同じコードをもっていて，局所変数が異なるだけである．そこでプロセスのコードはひとつにして，局所変数をプロセスの数だけもつようにすればよい．つまり，再入可能コードにすれば

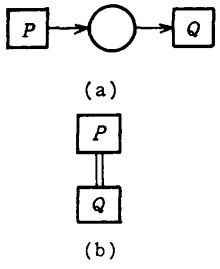


図-6　疑似コルーチン化

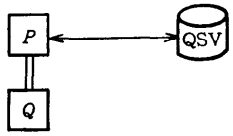


図-7　再入可能コード化

よい．プロセスを再入可能コードにするには，コルーチン化した手続きに対して局所変換（状態ベクトル）と進度指標を手続きの引数にする．

これを図-6 のように図示するが，通常状態ベクトルが大きくなり補助記憶が必要なので，プロセス Q の状態ベクトルを Q から分離して，それらをプロセス P が QSV で管理することを示している．

(5)のバッファによるスケジューリングは，入力データの管理をスケジューラがバッファを用いてせざるをえない場合の処理である．

(6)のプロセスの分割は，プロセスの複製を用意してその必要な部分のみを利用するようなケースである．

(7)の状態ベクトルの分割は，参照される局所変数（状態ベクトル）がごく少数のときそれらを分離しておき1次記憶に置くなどして参照速度を向上させるのに用いる．

3. 実　例

3.1 対象事象ステップ

例題の仕様記述から，対象の候補となりそうな名詞ないし名詞句を列挙すると表-1 のようになる．このうちから，着目する範囲外のもの，対象の属性や状態を示すもの，対象というよりむしろ事象であるもの，システムの機能や目的に係わるものを除かなければならない．また，洞察によって導入する「注文主」のような対象を追加する．

表-1　対象リスト

対象＼判断規準	採用	着目範囲の実世界外	事象の属性	事象相当語	対象の状態	機能目的
会社		×				
倉庫		×				
コンテナ	○					
酒	○					
銘柄			×			
倉庫係		×				
積荷票			×			
受付係		×				
出庫指示				×		
内蔵品				×		
空コンテナ					×	
出庫依頼				×		
電話		×				
在庫切れ					×	
数量不足					×	
出庫指示書						×
在庫不足リスト						×
コンテナ番号			×			
コンテナ数					×	
注文主	○					

洞察によって“酒を出庫させるものが存在する”と考える

表-2　事象リスト

事象＼判断規準	採用	同義語	機能	対象の状態
搬入する（コンテナ）	○			
混載する（酒）				×
受取る（コンテナ）		×		
保管する（コンテナ）	○			
手渡す（積荷表）		×		
出庫する（酒）	○			
搬出する（コンテナ）	○			
電話連絡する（倉庫係）			×	
記入する（不足リスト）			×	
知らせる（受付係，空コンテナ）			×	
出庫依頼を受ける			×	
出庫指示を出す			×	
発注する（注文主）（出庫依頼）	○			
出庫指示（酒）	○			

“注文主”による追加

また，事象についても表-2 のように候補となりそうな動詞あるいは動詞句を列挙し，同義語や機能に関するものや状態を示すものをとり除く．また「注文主」によって追加される事象もリストアップする．

3.2 対象構造ステップ

対象の構造を事象の時系列の視点から表示すると**図-8～図-10** のようになる．図で＊は反復を示し，○は選択を示す．

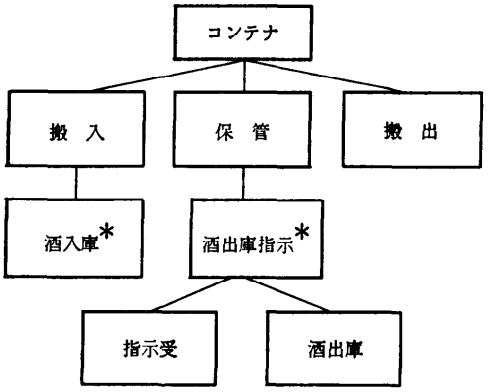


図-8 コンテナ事象構造図

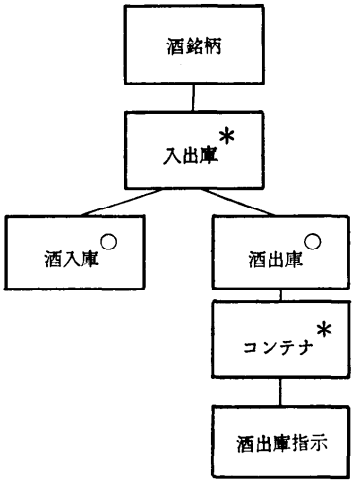


図-9 酒銘柄事象構造図

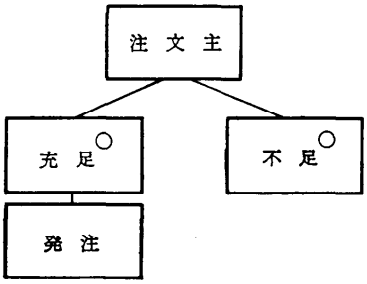


図-10 注文主事象構造図

3.3 初期モデルステップ

3.1, 3.2 でえられた対象，「コンテナ」，「酒銘柄」，「注文主」をプロセス化し，相互関係をデータの流れ結合ないし状態ベクトル結合で**図-11～図-13** のようにモデル化する．図-11 の破線より上が実世界であり，下がモデルである．「コンテナ０」，「注文主１」などの０は実世界，１はモデルを示している．矢印上の２

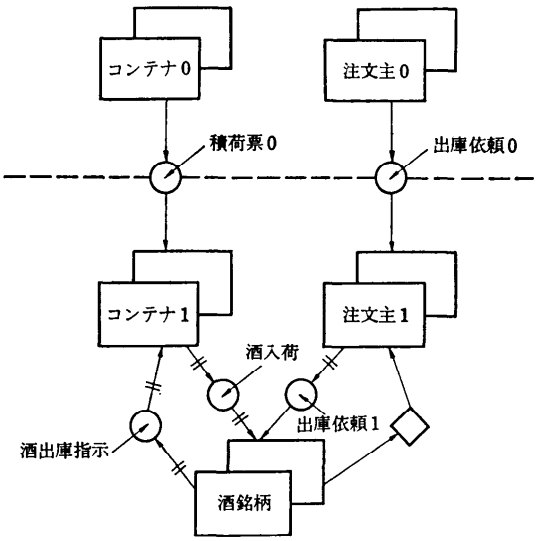
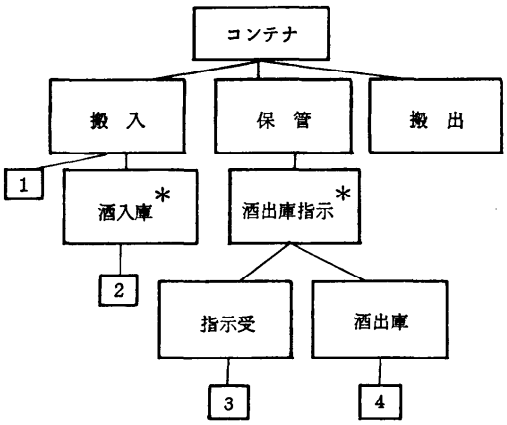


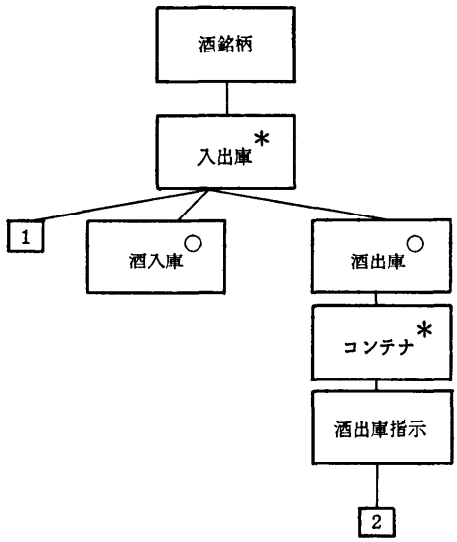
図-11 初期モデル図



1. read 積荷票０
2. write 酒入荷
3. read 酒出庫指示
4. write 酒出庫指示行 to プリンタ

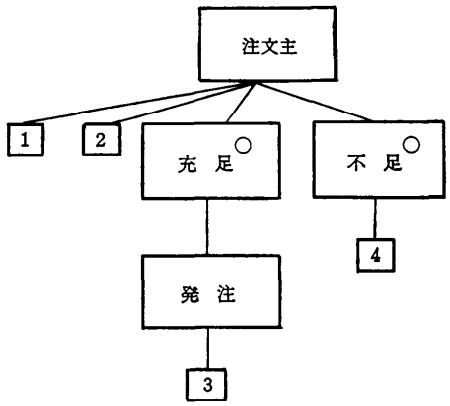
図-12 コンテナのモデル図

本棒は，複数のデータの流れ結合があることを示している．**図-12～図-14** に各モデル図，**図-15～図-17** にプロセスコードを書いておく．ここで，seq, sel/alt, itr はそれぞれ逐次，選択，反復的制御要素を示す．



1. read 酒入荷 & 出庫依頼1
2. write 酒出庫指示

図-13 酒銘柄のモデル図



1. read 出庫依頼0
2. get SV of 酒銘柄
3. write 出庫依頼1
4. write 不足依頼行 to プリンタ

図-14 注文主のモデル図

3.4 機能ステップ

例題で要求される出力機能は，在庫不足リストと酒の出庫伝票であり，これらを機能として**図-18** のように追加する．

```
コンテナプロセス seq
  搬入 seq
    read 積荷票0;
    酒入庫 itr
      酒入庫本体 seq
        write 酒入荷;
        酒入庫手続;
      酒入庫本体 end
    酒入庫 end
  搬入 end
  保管 itr
    酒出庫指示 seq
      read 酒出庫指示;
      指示;
      write 酒出庫指示行 to プリンタ;
    酒出庫指示 end
  保管 end
  搬出;
コンテナプロセス end
```

図-15 コンテナプロセス

```
酒銘柄プロセス itr
  入出庫 seq
    read 酒入荷 & 出庫依頼1;
    入出庫本体 sel (入庫)
      酒入庫;
    入出庫本体 alt (出庫)
      酒出庫 itr
        酒出庫指示;
        write 酒入庫指示;
      酒出庫 end
    入出庫本体 end
  入出庫 end
酒銘柄プロセス end
```

図-16 酒銘柄プロセス

```
注文主プロセス seq
  準備 seq
    read 出庫依頼0;
    get SV of 酒銘柄;
  準備 end
  本体 sel (充足)
  発注 seq
  write 出庫依頼1;
  発注本体;
  発注 end
  本体 alt (不足)
  write 不足依頼行 to プリンタ
  本体 end
注文主プロセス end
```

図-17 注文主プロセス

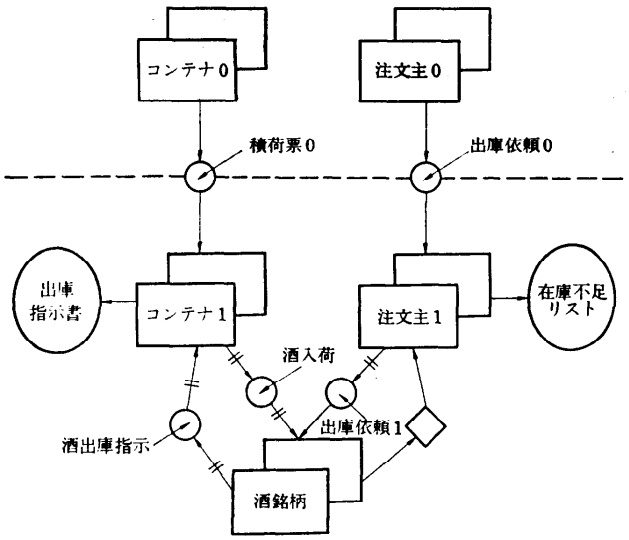


図-18 拡張システム仕様図

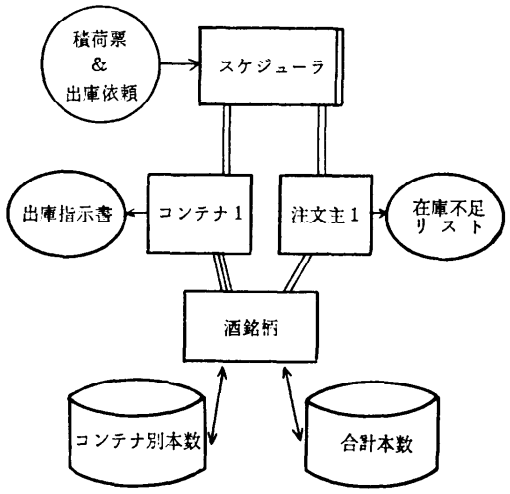


図-19 システム実現図

3.5 タイミングステップ

この例題では，とりたてていうことがない．

3.6 実現ステップ

ここで始めてスケジューリングプロセスを導入し，システム入力であるコンテナ積荷票と出庫依頼を受信(read)し，コンテナプロセスと注文主プロセスを疑似コルーチン化して呼び出すようにする．酒銘柄プロセスはコンテナプロセスと注文主プロセスに対する疑似コルーチンにする．

プロセス群は再入可能コードにしており，さらに酒銘柄プロセスの状態ベクトルに対しては，状態ベクトルを分割している(**図-19**)．

ここで，実現のテキストは省略した．

4. 若干の補足

4.1 ジャクソンシステム開発法の留意点

ジャクソン法はひとつの方法(method)であって方法論ではない．そこにある方法論的ルールの一端は次のとおりである．

(1) 難しい判断は(例えば，スケジューリング)後まわしにする．

(2) 誤り易い判断も(例えば，プロセスの転置，タイミング)できるだけ後まわしにする．

(3) 暗黙の判断はこれを避け，必ず明示する．

(4) 誤り易い判断は，できるだけ早めに検査をする(例えば，対象事象ステップ，モデルステップ，機能ステップごとに選択対象と選択事象をチェックする)．

(5) トップダウンによるシステム開発は，これを採らない(例えば，「機能」設計からシステム開発を行わない)．

(6) データ構造やデータベースは，ジャクソン法ではキーコンセプトではない．データベースは単に実現ステップにおける状態ベクトル値の集合にしかすぎない．

(7) 初期モデルステップまでユーザが参画すべきである．

(8) モデル仕様，システム仕様はプロトタイピングによくなじむ．

なお，初期モデルステップにひとつ注意を加えておく．実世界からモデルの対象への通信には，「誤り通信」を含むことが多い．そこで通常，データの検査を初期モデルに導入し入力部分系(input subsystem)を構成する．実例ではこの部分は省いた．

5. おわりに

この稿の実現(existence)は，畏友山崎利治さんの終始変らぬご指導に依っている．同僚中島敬一君の協力もえた．査読委員の方々からは適切なコメントを頂いたが，十分にはその意図をこの稿に反映できなくて残念である．心からの謝意を表したいが，よい謝辞が想い浮ばないのが心残りだ．なお，参考文献2)は近く邦訳される予定である．

参 考 文 献

1) Jackson, M. A.: Principles of Program Design, p. 299, Academic Press, London (1975).
2) Jackson, M. A.: System Development, p. 418 Prentice/Hall International (1983).
3) Hoare, C. A. R.: Communicating Sequential Process, CACM Vol. 21, No. 8 (1978).
4) 山崎利治：ソフトウェア工学，共立総合コンピュータ辞典第2版，共立出版 (1982).

(昭和59年7月25日受付)