この資料は，下記論文を著者河野が翻訳したものです.
Zenya KOONO and Hui CHEN, The Ultimate Systems Development Method Based on Finite State Machine,Proc of SOMET 08, p. 126-145, Oct. 2008.

有限状態機械を用いる究極的な開発方式

河野　善彌[a,1]，　陳　慧[b]
[a]*Creation Project*
[b]*国士舘大学*

要約：本設計方式は次の３基礎に基づく．１．設計が容易な約3状態の有限状態機械(FSM)モデルを使う．２．イベント駆動形OSを用いて，仕様レベルから最終的な実現迄の直接実行ができる．３．前記のFSMモデルで階層的にシステムを構成して，ソフトウエア規模を最小化する．本方式は高い生産性と高い品質を特徴とする．これは組込みシステム用であるが，如何なるシステムでも使える究極的な設計方式でもある．
キーワード：組込システム，システム設計，階層展開連鎖，小規模，ソフトウエア規模，文書，FSM，イベント駆動OS，SDL，自動設計

はじめに

この論文の目的は,組込システム向けではあるが一般のシステムでも使える究極的な設計方式を提案することである.

「組込システム」とは製品領域名の一種である．これは，以前はハードウエア製品であったが，現在はソフトウエアで制御される物を云う．例は「デジタルカメラ」である．昔は制御も含めてカメラは全て機械式であったが，現在では殆どのカメラはソフトウエアで制御される「デジタルカメラ」 である.

現在では組込システムはモデル駆動(MDA)構成の１利用で構築できると云う考えが標準になってきた．著者の見解もこれに近いが，より基本的であり，より広範である．両者は共に有限状態機械 (Finite State Machine, FSM) を基礎とするが，このFSMについては殆どのソフトウエア人が経験したことが無い．本論文は設計手順 を詳細に説明するので，MDA 方式と比較して理解する良い機会

[1] 著者代表 : 河野善彌 , 代表 , Creation Project, 〒 251-0875 神奈川県藤沢市本藤沢 2-13-5;
E-mail: koono@vesta.ocn.ne.jp

である.

２章は，組込システムの特殊性を説明し，本方式の概念を述べる．3章は設計手順を説明する. この設計は階層展開の連鎖を前提とするが, それは基本思想のひとつである，３．２では約3状態の単位要素的FSMの設計を説明する．ここではベストプラクテイスを示し, かつ後で説明する自動化に備える為に, 詳細に説明する．3. 3では，本方式の第２の基本思想である階層的なシステム構成を説明する．3. 4では本方式の第3の基本思想であるイベント駆動形OSを説明する. 説明はベストプラクテイスであり, 開発経験を積み重ねて得たもので, 大学の教育に利用して約100チームに演習させた．4章は適用結果で，5章は本方式の検討である.

2. 本設計法の設計思想

2.1 組込システム事業の特殊性

組込システムの特殊性については多くの議論がある. しかし, 未だに議論されていないことが一つある. 組込システム事業者は, 自由競争市場で事業を営んでいる.［1］（通常のソフトウエア事業者は，かような自由だが競争的な市場での事業の経験がない.）経営者は，競争に勝つ為に技術力を高めるべく，優れた人材を投入し, 研究開発に多大の経費を割く. また, ハードウエア技術者も各自の技術を高める努力を怠らず，夫々の領域での高い技術を形成している.

組込システムには通常のソフトウエアと本質的に違う所がある. 通常のソフトウエアでは入出力関係が固定しており，従って「組合論理」と云える．組込システムは通常（訳注 : 殆ど状態を持つ）ハードウエアを制御するので，*組込システムのソフトウエアは「順序論理」*（訳注：シーケンシャル論理）*である*.

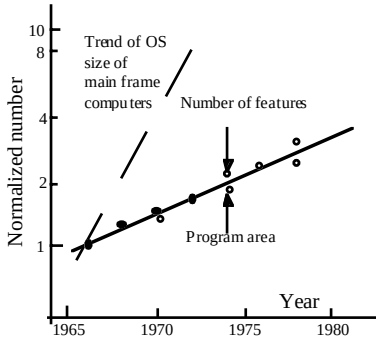

Figure 1. Increasing trend of software

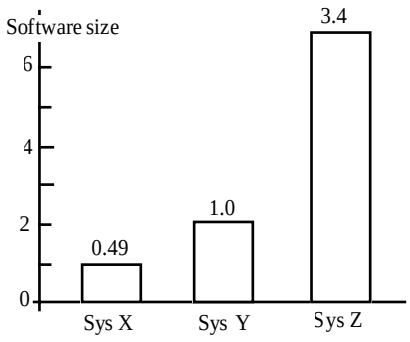

Figure 2. Software size

論理設計のどの教科書でも,「順序論理」部分は必ず状態を貯えるメモリを用いて構成すべきこと，その他の部分は「組合論理」で作れると，教える．ソフトウエアに於いても，全く同様である．組込システムのソフトウエアは，有限状態抽象機械 (FSM) を用いるモデル化を必要とする．通常の MDA では通常の OS を使用することとして，このソフトウエアの実現を自動化する．本方式では，イベント駆動形 OS を用い，仕様レベルから最終的なコードに至る迄，プログラムを直接実行できる．

技術推移を調べよう．図１[2]はソフトウエア規模とサービス機能数の増殖傾向を示すもので，世界最初の大規模な組込システムである No.1 ESS[2]の論文[3]から再プロットした．水平軸は年を表し，垂直軸は対数尺度でソフトウエア規模とサービス機能数を正規化して示した．両傾向線は同一で，約 10%/ 年の増加率を示す．この発見者河野は，続けて下記を見出した．・これはソフトウエアとかハードウエアに関わらない傾向である，・OS では更に高率の増大があるが，これはデータベースやデータ通信等の新分野の大きな母体追加と，以後の逐年追加から成立つ．NTT（当時電電公社）はこの発見に基づいて調査し，NTT 系通信ソフトウエアにも同傾向があることを確認した．これは一般的な傾向なのである．かくて 1990 年代（$1.1^{30} = 17.5$ 倍） には，世界の主要な交換ソフトウエアシステムの規模は数百万行に達した．

これらの数字は以下を物語る．

システム母体が大きいと，最初の開発コストが大きいのみでなく，以後の逐年の更新コスト（所謂メインテナンスコスト）が大きくなる．

ソフトウエアにとってソフトウエア規模は重要な要素である．図２[4]はある種の組込システムの制御ソフトウエアの中心部分の規模を示す．この差異は次のように云える．

＊ソフトウエア規模は Y で正規化すると，Z= 3.4，Y=1.0，X=0.49 である．この理由として考えられることは，次のとおり．

＊ Z は，発注者の要求により，この分野の技術者ではない人人で作られた．

＊ Y は，この分野の経験者に指揮されて開発された．

＊ X は，「規模の大きなソフトウエア」に悩まされた経験のある人達で作られた．規模を本格的に削減する為に「相互に独立な FSM で構成する方式」を採用し，各種の削減策を凝らした．

殆どの人の作業結果の特性は対数正規分布状にばらつくから，最小と最大は

---

[2] 1965 年に世界初のコンピュータ制御の電子式電話交換システム (No. 1 ESS) が運用を開始し，これは（旧方式である）巨大な電気機械式電話交換システムを更新するものであった．これは世界最初の大規模な組込みシステムであり，（通信の世界の巨人である）AT&T ベル電話研究所で開発された．同時期に（コンピュータの世界の巨人である）IBM は，システム 360 を出荷し始め，（コンピュータの大ユーザである）American Air Line は，世界初の座席予約システムの運用を開始した．これらは当時，世界的トピックであった．

平均値の1/3倍から3倍の広い範囲にバラつく．ソフトウエア規模は，その組織が競争的な市場で勝つ為に蓄積する技術能力の反映なのである．

（訳注：所謂IT系の世界でも，ソフトウエア規模に比例して作業工数が増すことは知られており，顧客の予算獲得や社内での開発計画等に使う．しかし，図２の如く明示的な研究結果は見当たらっｚ，その差異の原因も規模を制御する研究も公開されていない．開発現場の管理者は規模を抑制する意識を持つが，担当者層では「規模を抑制するより，規模は増えても早くに書き下し，作るが先決」の空気がある．IT系ソフトウエアの世界は商業主義に支配されている．何でもデータは非公開の雰囲気があり，原価に関る研究は公表されない．ハードウエアでの「研究の場は本音で実態を出すが，以後は絶対公開しない」雰囲気とは全く異なる．）

2.2 本開発方式の設計思想

本開発方式[1, 6]の設計のコンセプトは，以下の通りである．

最も大きなモチベーションは，競争に勝つことである．勝てるように組織を作り，勝つための戦略を立てる．通常の組織では，事業毎にプロフィットセンタを設ける．組込みシステムでは，勝つ為にシステム設計グループをプロフィットセンタに仕立て，後続するソフトウエア／ハードウエアグループの上に立たせ，製造部署にも圧力を掛ける．同時に，プロフィットセンタは事業に関する全責任を取らねばならない．従ってシステム設計グループは，R&Dに熱心で，同時に設計の自動化や生産の自動化に大きく投資するのが標準的である．

以下に戦略を記し，その主要点を本論文で説明する．

１．コストを最小化する．そのために（高い生産性）で（最小規模）のソフトウエアにする．

ハードウエアでは，（最も安価な部品）を（最小限に少ない数量）使う．これと同様に，（最小規模）で（最も作り易い）ことを旨とする．

＊この方式ではソフトウエアとは基本的なFSMであり，Midas[23]と名付けたイベント駆動形OSで，FSMを制御する．

＊FSMで最も設計し易いのは，約３状態の単位的なFSMである．

＊ソフトウエア規模を最小化するには，「組合論理」の場合と同じく，全システムを相互に直交的に構成すれば良い．

＊SEグループ以降でソフトウエアを開発するには，各種の設計自動化手段を使う．

２．製品のライフサイクルを通じて高い品質である[5]為に，

＊関係する全てについて，正確で正しい文書を残す．

＊設計は小さな進行段階毎に行い，厳格で厳しく正確にチェックする．

＊テストを合理的に実施する．

３．進歩する為の戦略

＊総て設計は革新させ続ける．
＊設計自動化を含み R&D を強化する．
＊同種の誤り／失敗を再度おかさないように強力にフイードバックする．
その他は後章で説明する．

３．本方式の中核技術

３．１　概念展開の繰返し

このシステムでの設計とは，意図的行動の為に基本概念から階層展開を繰返す原理に基づく．これは既にSOMET06に於いて公開した[8, 9, 10]所であり，人の大脳の理想化した機能モデルと看做せる．

図３[10, 11]は，「時計」プログラムの設計記録である．これには図式的手段と自然言語（学術および工学用語を含む）手段とを併用する．四角形の上下に半円形を着けた記号はデータを表し，四角形は機能を示す．データフローとは，

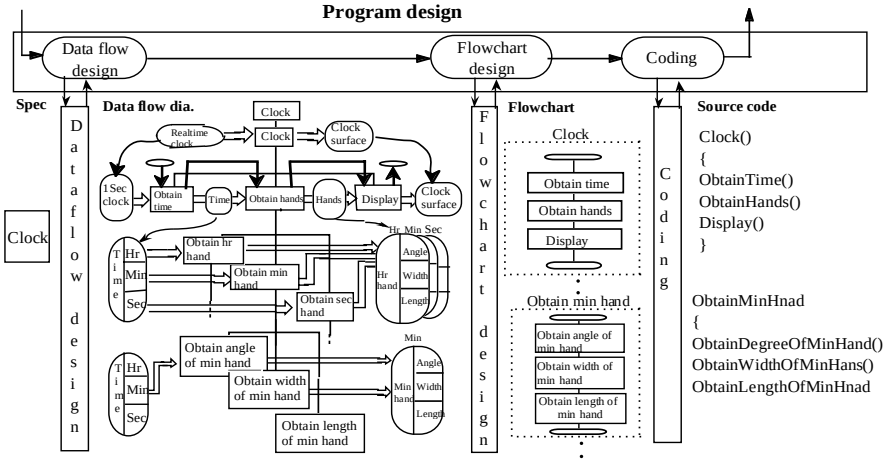

Figure 3. Design records of clock program

---

[3] Myers の STS 分割[12]．単位データフローを情報変換の連鎖である流れと見做す．これを，*最大抽象点(MAP, Most Abstracted Point)と名付けた，抽象度の高い*２データで流れを分断して，相互に独立な３部に分かつ事が，最適な分割である．入力側最大抽象点とは，入力の色彩を失なわないが，しかし，最も入力から遠い概念である．そして，出力側最大抽象点とは，出力の色彩を失なわないが，しかし，最も出力から遠い概念である．入力側は源泉 **(S** ource，中央は変換 **(T** ransform)，出力側は吐出口 **(S** ink) と呼ぶ．これら３者は相互に独立であるから，これはソフトウエア規模を最小にする分割である．Myersはこれを最高位置の１段階のみをSTS分割としたが，この方式では可能なら何処で何時使っても構わない．

データから始まり,機能次いでデータを繰返して最後にデータで終わる.左端のデータフローの最上部は仕様である「時計」である．この両側にデータを加えて，第２段は*親概念*である「時計」を表す*単位データフロー*になる.

これは階層展開されて下の詳細化データフローになる.これは*子概念*である「時刻を得る」「時刻表示を求める」「表示する」の３直列形[3]である.（ここで「時を得る」の記号の直上のひしゃげたビヤ樽状の記号からフローチャートが始まる.これは図の中央にも示してあり,これに対応するソースコードは図の右端に示した．これは制御の流れが必要なソフトウエアの特殊性である.）次に，先の子の中の「時刻表示を求める」を階層展開して今度は３並列な展開結果が示されている.

このように階層展開を繰返す内に，各概念は簡単になって行く．図の「分針を求める」例が示してある．各概念が充分に単位的になった所で実現手段/プログラミング言語によるソースコードに置換える.筆者等はこの原理を用いてソフトウエアの自動設計システムを開発して，SOMET06「8, 9」に報告した.

図４[7, 11]は,全体の操作を模擬した階層展開網である.図３では全ての階層展開の率は３展開である．しかし，最適な展開率は3 よりやや低く，e=2. 7182. . と思われる．図５は幾つかの実績例であり，図ａと図ｂはソフトウエア設計，また図ｃは論理設計である．著者等が既にSOMET07[11, 5]で報告したように，（ソフトウエアの）諸特性は人の特性が反映する．平均展開率もそれらの一つであり，その最適点は人にとって最善な点である.

これらは,階層展開の繰返しが意図的行動の為の大脳の機能モデルであることを示すもので，以下は設計のベストプラクテイスである[11. 5].

１. データフローを用い（必要ならフローチャートや等価な図面を加えて）自然言語で表わし，小さな進行段階毎に概念を展開する.

２．チェックのために，各詳細化の度に設計文書を記録し，残して行く.

３．小さな進行段階毎に，注意深く，厳密に，繰返してチェックする．チェックする対象は,小さな進行段階の初めに行ない,設計文書に記録したことについて行う.

本システムでソフトウエア規模を最小化する原理は，以下の通りである.

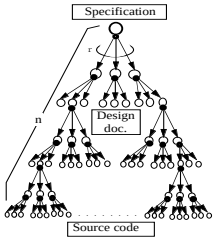

Figure 4. Hierarchically expanding network

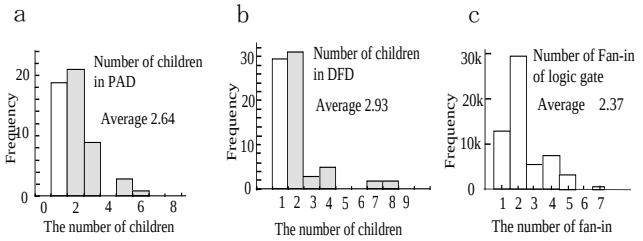

Figure 5. Expansion rate

１. 機能を相互に独立にする為に, 直列的に分割する. ソフトウエア規模は最小になる.

２. 共通部品化： 全て並列的なデータフロー群は, お互いに何処かで共通な所があり, 相互独立にならない. しかし, 何らかの共通部分があるなら, その部品群とそれを呼出す手続き群の如く相互独立に分割できる.

３. 定量評価と研究： 全て評価して設計の意思決定を行いつつ進め, 規模最小（論理的に最も簡潔）な方を選ぶ. もし, 良い案が浮かばないなら, それは後刻に研究課題として取組む.

ソフトウエア規模削減は, 時にはトラブルを招きかねない. 単なる規模削減のみを義務づけると, 判りやすさを犠牲にして規模を小さくする等の悪い方向に進みかねない. 殆どのベンダーでは, 規模は売上げの目安でもある. 彼らには, 規模削減は売上低減に写りかねない. これは人々の努力を損なうものではなく, 単なる労力的な作業重点から, より創造的な仕事へと云う共通的な理解に立つものである.

3.2 順序論理第１部, 単位的FSM

3.2.1 FSMの基礎概念

ここでは単位的FSMの「順序論理」を考える. 序論の後に, 単位的なFSMの例として自動販売機の設計を取上げ, 段階的な詳細化から複数FSMへ拡張する.

図６.aの交通信号を例に取り, 基本概念を説明する. ご存知のように, 信号灯は, 緑, 黄, 赤と一定間隔で移り変わり, 再び緑に戻って, 同一サイクルを繰返す. この３状態の*何れもが安定な状態*で, *夫々が排他的*であって, *全体が閉じたサイクル*を構成しており, これは*ある概念に対応する単位的なFSM*を構成している.

図6.aで, 各矢印はある状態から他の状態への遷移を表わす. これらは（状

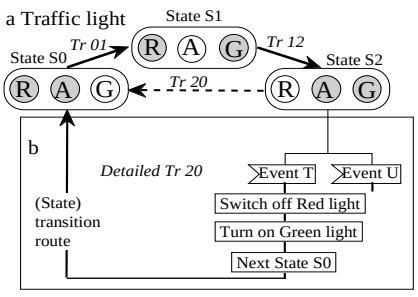

Figure 6. Traffic light

Table 1. Event (command) structure

Event		Event	Auxiliary info.
	Ex. 1	Coin inserted	Amount
	Ex. 2	Specify item	Item no.
Command Message		Command	Auxiliary info.
	Ex. 1	Change	Amount
	Ex. 2	Eject item	Item no.

態）遷移ルートと呼ばれ，ここでは$Tr_{旧状態新状態}$で表わす．この遷移の原因である入力は*イベント（遷移原因）*で，*スパイク状の信号*である．あるイベントの到来と共にある遷移が始まり，走行して終着点に至り，状態遷移を終える．

この方式では，入力プログラムが外部のイベントを検知すると，表１に示す標準パケットに編集する．通常の場合，イベント（出力のコマンドの場合にも）にはある付随情報がある．自動販売機では，例えば硬貨投入のイベントは「25セント」の付随情報を伴う．

図6.ｂはTr_{20}の遷移ルートをSDL[4]の記法で記した．ひしゃげたビヤ樽状記号（状態S2）は状態を表わす．その下の楔状記号はイベントである．SDL記法ではルートはフローチャート流に記載し，その中には単一あるいは複数の機能（例：処理）と単一あるいは複数の分岐を入れる．この遷移ルート中には遅延や停止を行わせないことにしている．Tr_{20}ではハードウエアの状態は状態S2から状態S0に変わる．イベントＴは遷移ルートTr_{20}を起動し，赤信号灯を滅火して緑信号灯を点灯する，これらは旧状態と新状態から誘導される．終わる前には「次状態S0」の定義が置かれる．このように遷移中の機能は初期状態と終着状態から系統的に得られる[15]．これらは「組合論理」部分であり，前記3状態と各遷移プログラムが「順序論理」部分である．

3.2.2 自動販売機

おなじみの自動販売機を具体化して単位的FSMにしてみよう．図7.ａは概要図を示すもので，IPO (Input Process and Output) 用紙に入れて示した．IPO用紙[16]はデータフローを書くのに最も適当な用紙である．図は，外部入力をI(Input)欄に外部出力をO (Output)欄に示した．中央のprocessingの代わりに図７の(Processing)P欄には，自販機の概要図を示した．

この図は，以下の条件を満たすように書く．

1. イベントを生じる全入力装置，コマンドで制御される全出力装置．この他にソフトウエアで制御されるハードウエアの状態に関わるもの．
２．システム担当技術者は該要図と関係するハードウエア図面を調べ，ハードウエアの各部を想起できるようにならねばならない．図は，読む人に関係す

---

[4] SDLはCCITT勧告である“**S**pecification and **D**escription **L**anguage”の略称である．ベル電話研究所のVaughanは研究モデルの交換動作をFSMモデルで表記した[13]．河野を含む日本の技術者はこの技術を各社の研究モデルに適用した．その結果として有用であることが明確になり，NTT（当時は電電公社）とメーカ群との間で交換システムの仕様記述に使用した．かくてこれは1976年にSDLとしてCCITT勧告になった[14]．CCITTは通信領域での国連の下部機関である．システムの仕様記述としてISOはLOTOSを推進し，CCITTはSDLを推進した．SDLは産業界で広く使われており，1980年代には，図式とテキストの両表記ができる形式仕様記述に成長した．これはUMLに技術移転され，UMLはVersion 2に進化した．

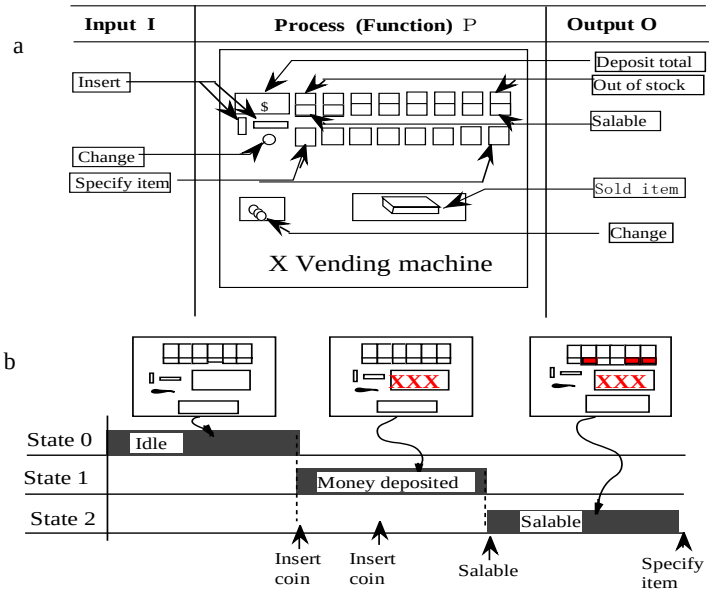

Figure 7. Vending machine

るハードウエアを思い起こさせるものである.

ソフトウエア人が色々な視点を持つが，システムの最上位のFSMを決めるためには以下に従う.

＊*システム自体に焦点を当てること,*

＊*システムを最も抽象化した振舞いを示すものを取上げる,*

お金を投入するとその指定されたモノを出力することを具体化して,

＊*各状態で関わるハードウエア資源の状態が夫々指定できること,*

「自動販売機（システム）X」と云う名前に

＊*該FSMの概念は，与えられた名称に対応するものにする.*

二つの着眼点がある. 一つは自販機自体であり，他は販売される商品である. 全体システムに関わるから，自販機自体がベターである.（コンベヤーシステム等では，本体システムと搬送される対象の両者が必要な場合がある.）

FSMの状態を定めるには，これら３状態がハードウエアとソフトウエアの両者の最高レベルのインタフェイスを規定することに留意する. 顕著で特徴的な所に注目する. 図7. bはタイムチャートで３状態の視点を示した. 図に付けた小さな図は，設計者が注意を払った諸点を示す. 最初は初期状態である. 次は投入累計額表示が表示される状態である. 最後は，単数または複数の商品が販売可能である状態であり，これら各状態は安定で，相互に排他的であり，かつ全体として閉じた系を構成している.

これは教育用であるから，幾つかシンボル図形が付加されている. 厳密に云えば「状態を定義する図面」は，もし必要ならその関わる状態毎に書くべきであ

る. 状態毎に関わる全ハードウエア資源を洗出し, 状態毎に関わるハードウエアの状態を明記せねばならない. 初期状態と遷移後の終着状態で定義されると, 遷移ルートのプログラムは両者の差を実行することに帰着する[15]. もし簡単化できるなら何らかの抽象化したシンボル図形表示が望ましい. しかし, 簡単だからと言っても, それは漫画ではなくて, ハードウエアとソフトウエアの*公式の仕様書*である.

次は全入力を定義する. FSMの名称（概念）, 状態およびイベントが相互に辻褄があうか調べて見よ. イベント（あるいはハードウエアに向けて送出されるコマンド）は, 表１のように構成が標準化されている. 付随する情報は,「組合せ論理」部分の主流であるデータであることがある.

これらは, イベント（コマンド）の仕様書は, *公式的な契約あるいは合議結果の文書であり*[5], *単なる辞書ではない*. 総覧的な図表で全体構造を示し, 出来るなら図式的な（例：表１）展望を与え, 階層的な表には各イベント（コマンド）および付随情報毎の定義を記す. 各定義は, データ名称の定義から始まり, 下位のフイールドの定義へと階層的に展開する. これらは多くの人々が使うから, 各記述は正確厳密明解かつ簡潔でなければならない. これらは法律の条文のように正確に書かねばならない. データの長さ, 型式, 初期値等や, 占有データ量等も記述する.

この仕様書がソフトウエアグループに引渡された後, 前記情報とシンボリック名称が自然言語記述の名称の後に記載される. ソフトウエアグループ以外の全員は自然言語を話す世界なのであるから, 外の世界ではシンボリックなコードネーム等は使用を禁止する.

---

[5] 全グループの品質を高める為，誤りにより後続部署で損失が生じた時には，該誤りを作り込んだ部署にその損失を課することをお勧めしたい．その正邪の判定は関係する文書に基づいて行う．各部署は，損失に備える予算を予め与え，経営者は各部署の累計損失／予算（百分率）で管理する．各部署は自己損失と同様に，関係部署に与えた損失を減少させる努力を払わせる．歴史的にハードウエア職制は，これらの厳しいルールと継続的な改善活動で向上してきた．ソフトウエアでも同様なアプローチで取組ませるべきであろう．

訳注：　工程の上位部署から流される図面の記載や業務に関わる情報は，後続部署に対する公式的な仕様～命令である．従って必ず従わねばならない．後続部署では，この「公式的な仕様～命令」に背いて作業する時には，当該「公式的な仕様～命令」を執筆／作成した人に相談して，その後で原則的に当該部署の責任ある管理者の許可を得て「変更する」あるいは「変更する仕様書」を発行して貰わねばならない．産業界では全てにお金が付き纏う．従って責任と権限の所在を明確にする．更に，同様な明確化の為に，必ず文書化するとか責任者の日付印を捺印する等を行なう．正邪は基本的に必ず文書により，作成～表記する規格～基準に照合して行う．これは一見型式的～不人情に見えるが，公的に罪刑法定主義を取ることが近代以降の国家方式であるのと同様である．逆に言うと，前記のような責任と権限の所在が不明で，文書等が不明な世界は近代以前の蒙昧な世界と謗られても弁解のしようもないのではなかろうか？

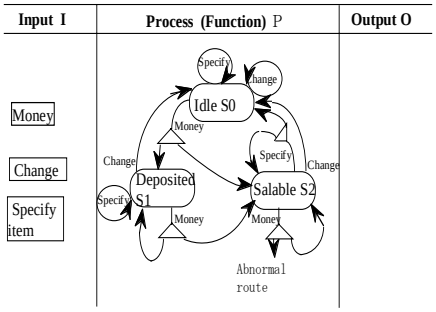

Figure 8. Transition route with decision

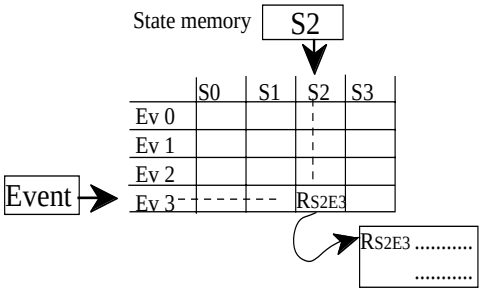

Figure 9. State-event matrix

3.2.3 遷移ルート

次の第一段階は，図９の状態遷移原因表を参照しつつ，図８[6]の単純な状態遷移図を書く. 状態遷移原因表とは, 全ての状態と全てのイベントを記した２次元表である．ある状態（例 : S2)とイベント（例 : Ev3）の交点に，遷移ルート名を記録する．ここでは例えばTr_{S2EV3}である．単純な状態遷移図を書く目的は，遷移ルートのみに注意を集中する為である. 実際には, 出発点近くにイベント名を記し，遷移ルートには遷移ルート名を記す．今は簡単の為に省略した.

最初に図７の該要図を見て, I欄のハードウエア入力を調べてイベントを見付ける. イベントの仕様を参照し状態遷移原因表を指している状態と併せて, その遷移ルートの名前を決め，状態遷移原因表の交点欄に記入する．また，図8のように何の機能も持たない単純な遷移ルートを描く.

ある遷移ルートは，複数の終着状態を持ちうる．この場合には遷移ルート中に分岐を挿入する．何の機能も果たさない遷移ルートは，その状態から出発し，再びその状態に戻る円を描く. 状態遷移原因表の各欄が全て埋まったら, 終りである. 各ルートが付けられれば, そのシステムは何があっても動作できるシステムになる.

図9はイベント駆動OSの原理を示す．OSは各FSMに対応した各状態記憶を持っている（例：Ｓ２）．あるイベント（例 : Ev3)が到来した．状態遷移原因表を用いると，遷移ルートはR_{s2ev3}と求まる．そしてOSは下に記したR_{s2ev3}に対応するプログラムを実行する．これは終了前に，次の状態の定義情報を持っている．OSはこのFSMの状態記憶を更新する．これは大変に簡単で，仕様のレベルから最終製品のレベルに至る迄，直接実行が可能である.

この方式では, 誤りを防止する為に遷移ルート中には遅延も中断も許さない. 必要なら中断の為には, 内部状態を設けても差支えない. 動作再開の為にイベント等を用意する. タイミングの為には，タイマープロセスを設け，これにタイミ

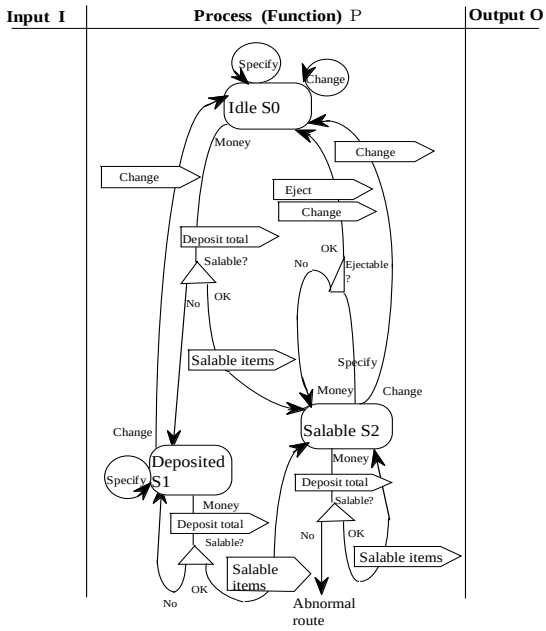

Figure 10. Output inserted transition diagram

ング情報を送り，その設定時間終了後にタイムアウトのイベントを返送する．

第２段階は，各遷移ルートに図１０に示したように，全出力を挿入する．状態の差から必要なハードウエア駆動コマンドが生成できる．コマンドは即答可能な応答（例：動作不能，OK/NOK 等）以外１方向性であり，高速に短時間で実行できる．図８に示した簡単な状態遷移図のコピーを作り，この第２段階の作業に使う．誤りを避ける為に，書直しはしてはならない．このコピーの図に，全出力を挿入していく．このようにして得られる第２段階の図は，厳密にA4用紙１枚に収める．（もし複数頁に跨がると，誤り率が「増える．人は無意識の内に誤るのであるから，出来ることは誤らない各種の策を打つことしかない．）

設計とは，ある入力文書から詳細化を行ってその出力文書（詳細化部分のある場所に）に記録することである． *誤りはこの両文書の間で生じる．後続するチェックでは先行した設計で行った作業結果を厳密にかつ厳格に調べることである[17, 18]．* もしハードウエアに関わる所があるなら，システムの内に留まるのではなく，ハードウエアの内に一歩踏み込んで関係する個所を調べる．（例えば，ハードウエアセンサーとその特性等）

図 10[6]に示すように，元の単純な遷移ルートに，杭状の出力を入れて行く．その原理は既に図 6. b の信号灯の例で記した．

1. 各状態のハードウエア状態を示す図を準備する．
2. 以下の手順で全遷移ルートの作業をするが，図面は *A4 用紙 1 枚*に収める．
3. ある遷移ルートを取上げ，ハードウエアの状態図から前後の状態の　差を明確にし，必要なコマンドを挙げて，その遷移ルートに挿入していき，詳細化するが，厳しくチェックして，原本と後続作業用の図面を作る．

コマンドが遷移ルート中に沢山入るなら，一連のコマンドを送出する関数を設け，遷移ルートでは「これら関数をコールする」のみに留める．全出力の挿入が終わったら，全遷移ルートで（ハードウェアを含め）全状態が出来上がる事を机上チェックで確認する．

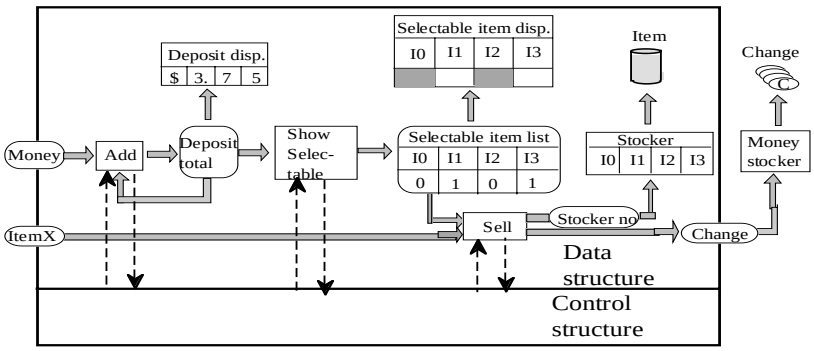

Figure 11. Data flow of a vending machine

訳注：上記のようにA4用紙1枚の図面を中心に，このFSMに関わるハードウエア関連の全動作が事前確認できる．ルート自体，ここのハードウエア動作，次の内部処理で基本的な全機能の基本の設計が行われ，また事前確認が行える．それ以降は，個々の遷移ルート中心になる．面倒ではあるが，小さな単位ステップ毎に設計して文書化してきた．ここではこれらを活かして，この機会に厳しくチェックして，本質的な設計の大きな誤りを防止しようとするものである．

3.2.4 FSM 設計を仕上げる

あるFSMの設計を仕上げる最後の仕事は以下のとおりである．

1. （次状態定義を含め）必要な機能を挿入する．
   ハードウエア駆動出力条件が挿入されると，内部処理が設計され機能が挿入されねばならない. 先の図10の遷移ルートに新しい関数が追記される. 人の思考力は注目する対象の上下10シンボル以上は思考範囲を超えるから，階層的に関数を設けて各ルート中の機能の数を減らし, 常に全体を表わす図は常にA4用紙1枚に留める．前記同様の配慮を払い，同様に厳しくチェックする．

システムレベルのデータフロー設計は, 内部処理の要求条件を最も良く示す. 図11の上部は，*データ構造*と呼ばれ，自販機のデータの流れを左端の当初入力，投入金額累計，販売可能な商品のリスト，右端に最終出力で示したものである. 図11の下部は，自動販売機の*制御構造*と呼ばれる．両方向の矢印は，遷移ルートプログラムがデータ構造に関わるプログラム（例:「加える」とか,「販売可能商品を表示する」）を呼ぶ様子を示す.

ある種の応用（例: デイジタルカメラ）では, データ構造がそのソフトウエアの大部分を占める. システム設計者（主任設計者）はカメラの製品企画のエキスパートと当該品を担当するソフトウエアグループの人々を呼び, 該カメラのデータフローを階層的に描く. 各部分に分解すると, カメラの各種機能とデータとの関係, 内部機能（データフロー）, FSM群との関りが明瞭に浮かび上がってくる.

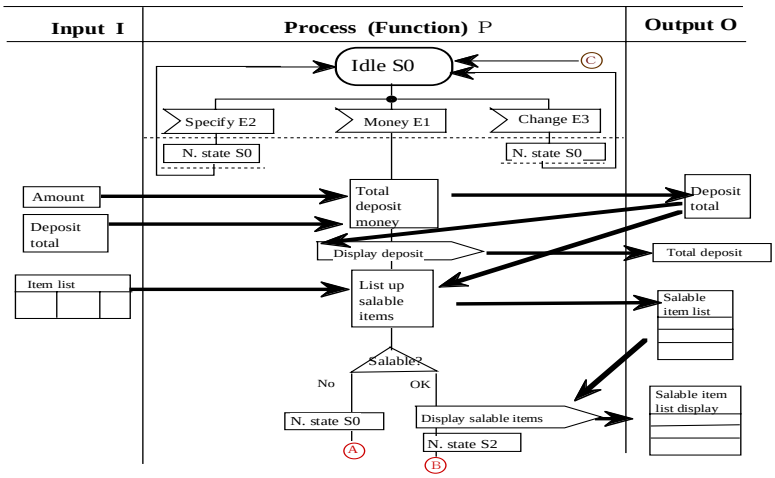

Figure 12. State transition diagram in SDL expression

中心構造であるアーキテクチャが決まり，データ仕様が定まる．これらの間，ソフトウエア規模を最小化することを怠らない．*制御構造*の大部分はシステム設計者達が設計した FSM で実現される．

2. 出来上がった状態遷移図を SDLCASE ツールに移し変える．
   これは転写であるから，注意深く厳しいチェックを行う．図 12[6]は，自動販売機の状態 S0 での状態遷移図を示す．*図は均一に，明瞭に，バランス良く，美しく描け．これを上手に出来る人が良きシステム設計者に成れる．*かくして出来上がった SDL の図は，憲法の１条とも云える．

図１２では，IPO 用紙の P 欄に SDL 図が書かれる．図に見られるように，I 欄にデータ，O 欄のデータと矢印を描くと，遷移ルート中の処理操作が容易に理解できる．これは元ハードウエアの人や設計自動化システムによる設計やチェックも容易になる．設計の標的（即ち，約 3 状態の FSM）は標準化されているから，設計手順やチェック手順もまた，同様に標準化できる．自動操作を順次取入れて行けば設計自動化システムも自動化の度合いを向上できる．

3. イベントで動作が始り次状態定義で終わる各状態遷移ルートを，各状態遷移関数として定義する．
4. 構造化チャート CASE ツール[6]，加えてできるならデータフロー CASE ツールを用い自然言語で階層的構造的な設計が出来る．

---

[6] 構造化チャートは，プログラム表示に適しており，最終段でソースコードに変換する．従ってこれは小さな段階毎の詳細化を可能とし，また設計者は進行中に設計内容を容易に確認できる．これは日本の高品質ソフトウエアを可能にする戦略兵器である．大きな流れは PAD(**P**rogram **A**nalysis **D**iagram，日本語版URL：http://www.hitachi-system.co.jp/topital)と HCP(**H**ierarchical **C**om**P**act Chart, 日本語版 URL：URL：http://www.denso-create.jp/service/products/nnheadway/index.html 及び http://www.oki.com/jp/Home/JIS/New/OKI-News/1996/11/z9646.html）である．

5. 充分に机上チェックした後に，全関数について，プログラム仕様を確定する．

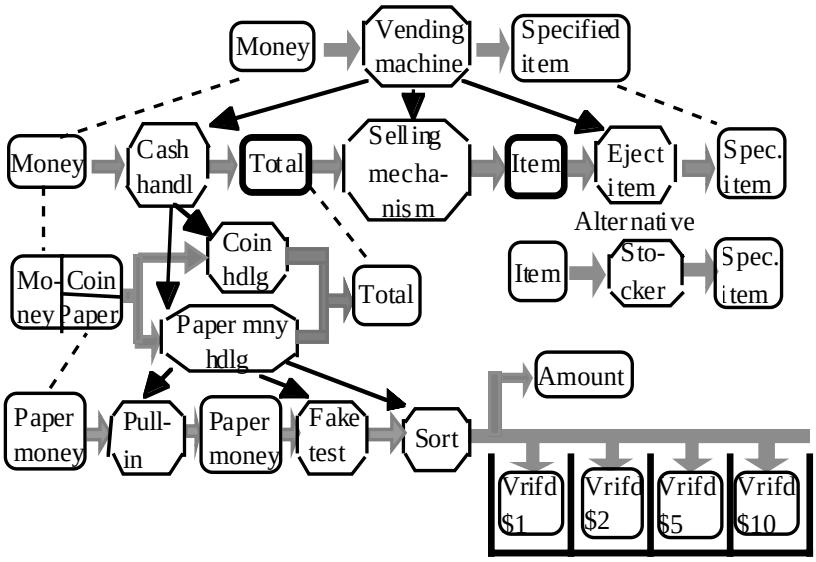

Figure 13 Vending machine with hierarchical FSM group

6. かくして定義されたスケルトンの関数にスタブを付けて，システムを実験的にシミュレーションする. これはシステム設計の作業結果の品質を立証する為に行う．

3.3 順序論理　第２部，複数FSM

本方式は，相互に独立な約３状態のFSMを階層的に構成して系を作る．図１３[6]は自動販売機をこのように構成した例である．人の脳裏では，これは図３のように投影されている．

この図を見ると，各FSMは約3状態で設計でき，レベルを降るにつれ，レベル毎に全く違う概念レベルであることが理解できよう. 更に, 各レベルで殆どのフローが直列的だから各FSMが相互に独立であり, 従ってソフトウエア規模は, 狙いとおりに大幅に減少できる．これらから，だれでも容易に設計できる．

図１３の最上部から設計を見よう．最上位の概念「自動販売機」は，典型的なMyersのSTS分割であり，「金銭収受」，「販売機構」，「排出機構」に分解され，各FSMは相互に独立である．

オートマトン理論によれば，入力側FSMがx状態，内部処理FSMがy状態，出力側FSMがz状態を持つなら，全体系はx・y・z状態を持ちうる．今仮にx =

y = z = 10 なら, x・y・z = 1000 である．コストは状態数に比例するから 3 FSM では 30 状態であり，その比は 33:1 に達する．これは仮定条件での比較ではあるが，大幅な規模の減少が理解できる．

次のレベルでは，「現金収受」は，図 3 の第 4 段に於て「時刻」から「時」「分」「秒」への展開と同じく，入力である「お金」が「紙幣」と「硬貨」に階層展開するに併せ，「紙幣収受」と「硬貨収受」に階層展開される．次のレベルでは，フローは殆どデータフローであり，STS 分割を使えば直列に展開され，相互に独立な FSM 群になる．従って，様相は図 3 と全く同じになる．

次に「お金」が「紙幣」と「硬貨」に分かれる場合を考える．この他に同様な概念群がある．

・キャッシュカード，クレジットカード，電子マネー，ポイントやトークン

・日本円，韓国ウオン，中国ユアン，米ドル, . . .

「販売機構」や「排出機構」でも同じ事態が起こる．これらの拡張が最初から想定すれば，変更は最小に留められる．これ以上の説明は最早不要であろう．ハードウエアでは，かような取組みは標準化に基づく戦略的な事業計画として周知である．

ソフトウエア生産性はホットな研究テーマである．研究が必要である．

3.4 Midas OS

著者のひとり河野は1975年に高度に多数のFSMを同時動作させる為にイベント駆動形 OS を作った．それを使うと使用者に色々な御利益があるので，ギリシャ神話の神「マイダス」と名付けた．これは触る物を金に変える力を持つと云われる．主たる機能は FSM の制御であり，独立の OS としても使え，ある既存 OS の下にぶら下げることもできる．以下に構造を説明する．

図 14[23]の右には各種 FSM の為の状態遷移プログラムがある．左端にはイベントを検出する入力プログラムがある．その出力は，底部左に示した標準化したパケットを使う．（プログラムは，自分の後のプログラムに最適なように考えて

pdf 用
仮図面

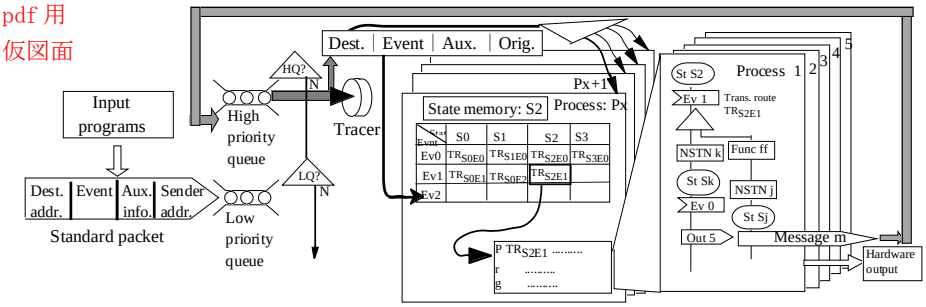

Figure 14. Midas OS

設計するべきものである.）それは，あたかも封筒とおなじように，行き先アドレスと発送者アドレス（例：FSMや，プログラムあるいはハードウエア装置のアドレス）を持ち，手紙本文のようにイベント（コマンド）と関係する付随情報を持つ.

高優先度と低優先度の２レベルのキュ-を設ける.入力プログラムからのイベントのパケットは必ず低優先度キューにつなぐ．一方FSM間通信用のメッセージ/パケットは全て高優先度キューにつなぐ．これは複数入力による混乱を防ぐ.パケットで標準化した御利益により,全体に共通で大容量の空きキューを共用する．これは各個に空きキュ－の容量計算をする手間が省ける.

プロセッサーが空きになると,まず高優先度キューを走査する.もし待合わせているパケットが何も無いなら,低優先度キューを走査する.走査中に待合せているパケットがあれば，OSは当該パケットをキューから取外し，図に上右に示したように，パケット中の宛先から指定されたFSMに到達する．このFSMの現在状態記憶から現在状態を得る．パケットからイベントを得て,（訳注：遷移プログラム対応に書直してある）状態遷移原因表から,この状態でこのイベントに対応する状態遷移プログラムを知り，図の右に示したSDLのプログラムを実行する．もし（白杭状記号の）出力があれば（直接，あるいはしかるべきキュー経由で）送出する．各状態遷移プログラムの最末端には次の状態番号（図ではNTNX）を本FSMの状態記憶に設定する.

高優先度キュ-が全て尽きたら,低優先度キューを走査して,新しいパケット(外部からの新しいイベントを含む）の処理を行う．ここに記した処理は高速に実行するように設計する.

訳注：　遷移原因に表記された他の信号との論理積を取り,状態遷移の条件を絞る方式がある.この実現方式はOSの処理時間の増大を招くから,絶対に行ってはならない.イベントを送出する時,後位で最適なように配慮する原則に立脚して，前位で必要な条件に絞り込む．それが出来ないなら，受けた側で必要な条件に絞りこむ．一般にはソフトウエア規模最小が基本鉄則であり，OSの動作に関る場合には，実時間損失最小が鉄則である．単純にプログラム内で閉じない異常/障害の場合には，原因切分けが必ず面倒だから，必ず対抗処置を設計に織込む.

終りが無いリング状記憶が設けてあり,パケットが処理される毎に関係する情報を記録する.システムが停止すると,前記リング状記憶から,これら先立つ幾つかの状態遷移プログラムの軌跡が取出せる.ソフトウエアおよびハードウエアの殆どの異常は速やかに原因が分かる.この為に状態遷移では一切の遅延や中断を許さず(訳注:ハードウエア障害に対抗する為あるいはプログラムレベルに関るタイマー等の割込みでの中断は除く),*FSM間のデータや情報の受渡しはイベント（コマンド）パケットで必要に応じて行う.*

訳注：このトレーサー機能を付け，常に実体に対応する最新の状態遷移関係図面を用意し，これらから問題を切分ける能力のある人を配置したこのプロジェクトでは，システムレベルでテストでは異常検出後平均4.5時間で異常修復を完了できた．現在なら更に早く処置できる.

タイマープロセスは依頼元FSMに「タイムアウト」イベントを返送する．もっと高速が必要な場合には，割込みレベルで動作する別ユニットを設ける．

高速を求められる実時間システムでは，キュ－の長さは短くする．（訳注：高速プロセッサで実行時間を短くする，更により高速なプロセッサを用いて，軽負荷状態で働かせてキューの平均待合せ時間を短くする．）高信頼度システムでは，部分的～全面的なデュアル／デュープレクスが行える．この部分はOSシステムの中核部分であるから，如何なる組合せも可能である．本部分を既存OSに組込みフルイベントドリブンなOSシステムが作れる．これら単純で系統的であり，厳しいルールで統制したから単純で高速になり，透明度が高く信頼度が高いシステムができ，かつ（処理能力，各種時間特性，異常事態等）評価が容易なシステムが作れる．ソフトウエアの人には厳しい拘束条件に見えるが，（訳注：キチンと理由を付けて定量的にお話して協力を求めると），ソフトウエアの人達は喜んで従って呉れ，「制御部とはどんなものか，始めて分かった」と云って貰える．

4. 適用実績

本方式の適用実績として，著者等の埼玉大学での教育[7]を報告する．これは10年余り（約500人）100チームへの教育記録[6]で，３年生のソフトウエア工学１月間のチーム演習であった．作業方式は本報告のとおりである．

３年生は初期教育からソフトウエア人に習熟する過程に居る．まだプログラミング能力は低く，作り込み欠陥率は100誤り／k行である．チーム当り平均工数はおおよそ150時間であり，70%が設計で残り30%がテスト工数であり，机上チェックにより誤りが事前に摘出されたことによる．

Table 1. Quality reports (1995)

Summary of quqlity report (1995)					
Team	S/w size	No. of defects found T1/2	T3	QA	Defect Intensity
A	283	0	0	1	3.53
B	438	4	4	1	20.5
C	486	3	0	2	10.0
D	357	2	2	1	14.0
E	858*	3	0	7	11.7
F	495	1	7	2	20.0
G	442	4	1	1	13.6
H	463	3	0	0	6.5

* For Parking tickets and outgoing control
T1: Test for each function T2: FSM
T3 and QA: System test including running
Defect intensity is for Kilo lines

[7]埼玉大学情報システム工学科では，２年間の初等的プログラミング教育の後，３年生はC言語，OSやソフトウエア工学を学ぶ．著者等は組込システムの開発の概要教育の後，FSMの原理，イベント駆動OSであるMidas OS，単一FSMで３状態の自動販売機とそのプログラムの教育をする．１学生チームは4～7人である．教師側でリーダーシップがありプログラム能力の高い人を選んでチームリーダに，次にプログラミング能力でOS(Midas OS)スペシャリストを選び，その他はバラつかせて全チームが同程度の力であるようにチームを作る．課題は複数FSMで任意ではあるが，トップセールになるような自販機類を１月後迄に作り上げる．（皆が決められた役割を演じる）ロールプレーイング方式である．リーダは社長であり，全員を統率し，チームの成功の為にその他の全員が協力する．終了後に，リーダはチームを代表して紹介し，その他の全員は各自の業務を報告する．1991～2001の間は全規模で，次の５年間は規模を削減しておこなった．

表２は1995年の各チームの成績一覧である．ここでチームEは２システム（駐車券発行システムと出場管理システムの両者）を開発した．他は全て3FSMで開発した．左端を見ると，チーム当りの開発規模はC言語で487行であった．従って10年間の標準としてFSM当り150行程度であり，正味は約100行である．

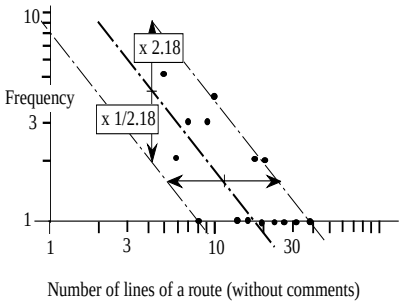

Figure 15. Route length distribution

余分なコードを書かないように，チームリーダが指導したチームの，より詳しい実績を図15[6]に示す．図は遷移ルートプログラム行数の分布を示す．横軸は行数，縦軸はプログラムの頻度であり，両対数尺度で表示している．プログラム規模は5行から40行の間に分布している．（訳注：このように個々のプログラムは短いから）OSにより高頻度で走行するプログラムは「速度優先」に設計せねばならない．かような単純さからプログラムに起因する誤りは大幅に減る．この場合には，データ構造部分の負担は無い．プログラムを階層構成で作る時には，自然言語表記部分を極力大きくしてやり，プログラムを図15のように小さくする．

図には帯状領域[8]を形成する．太い破線の傾向線は打点群の中央を貫いており，これと等距離の２本の副傾向線を引いてある．これら傾向線は，負の指数状減衰[9]の分布を示しており，上の副傾向線は中央傾向線の2.18倍上に，下の副傾向線は中央傾向線の1/2.18倍下にある．これらは遷移プログラムの一定率のバラつき特性を示している．脚注８と９には詳細を記した．

次に品質を示す．表２で，テストT1はC言語の全ての関数を全数テスト，T2はFSM単位のテスト，T3はFSMを順次統合してシステムレベルでテストしてテストで摘出した欠陥数を記してあり，右端は全摘出数を規模で除した平均欠陥密度である．この年度では，開発終了後に文書チェックから実動作試験迄を検査専門家が行う「品質評価」を行わせた結果も含めてある．全体平均は12.8欠陥／ｋ行であった．

図16は他年度の記録である．水平軸はテスト摘出欠陥密度，平均欠陥密度は約12欠陥／ｋ行である．これら実績の示すように，平均値は作業に対する環境を含む拘束条件の反映である．学生の作込み欠陥密度は100欠陥／ｋ行である．彼らは通常かなりの割合の欠陥をテストで摘出する．この開発では80%以上（訳

---

[8] 統計学によれば対数正規分布なら，平均値を中心として×３σ～１σ/3の帯状になり，これは多数の小さな要因が相互に積の形で作用して生じる．

[9] トラフィック理論によれば，生起確率が小さくランダム性なら，その長さの分布は負の指数減衰状になる．（訳注：アセンブラプログラムであるレジスタに収容されたデータが，設定されてから使用が終わる迄の時間長さは負の指数減衰状であった．これに倣って分布形状を調べた結果である．）

注：100−12=88）が気付かれない内に事前除去されている. 従ってこの品質の高さは学生に強い印象を与える．報告には，「オーッ！組合せたら一発で動いた！」等と報告してくる. 図16は品質と生産性の関係を示すもので，垂直軸は生産性で，各打点はチームに対応する．図の小さいバラつきは，

１．対象が全て約3状態のFSMである．

２．工程は厳しく統制されている．

３．メンバ学生は各チーム略同能力．

現場ではこのような明確な傾向は見える機会がない.

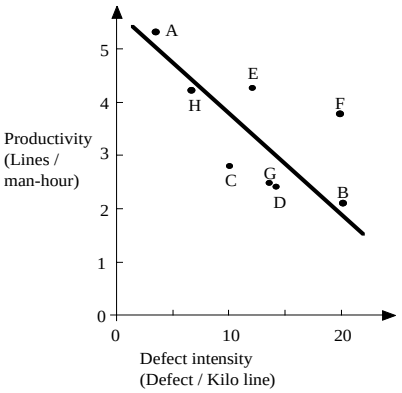

Figure 16. Productivity and quality

図16の傾向線は，欠陥密度が低い程，生産性は高い，ことを示す．品質も生産性もそのチームの知的能力に関るから, 図の傾向は当然の帰結である. *品質の悪い組織は同時に生産性が低い組織である. 実行する工程を良くしていけば, 品質も生産性も向上する.*

5. 検討

筆者等は，システム設計技術者の立場から,「モノ」と「プロセス」について，更に「特性」について詳細を報告した．ソフトウエア規模は最小化されるから，彼らはプログラムと云う重荷から開放され, システム設計が要する所に, 時間と勢力を注げる．各自の仕事には，多くの研究論文，調査報告，各種の比較結果，討論，各種のトレードオフと各種の準備を要する．この集積が仕事に反映する．

本方式は, 微小粒子状に簡単で, 厳しい規格に従った図面化がされるから，これで開発されたFSMやシステムは，市販可能な部品になりえる．このソフトウエアの広範な使用は産業界を変えうる. このシステム部署に後続するソフトウエアグループの人々にとっても事態は同じである．もはや「ムダ」「ムリ」も「ムラ」は存在しないから，SOMET07[18]で説明した「合理的定量的科学的なソフトウエア工学」に従い開発すれば良い．かくて，彼らは資格ある技術者に成れる．ソフトウエアグループは単に生産と，各製品の価格，品質，納期に責任を持つのみでなく，SOMET06[9]に報告した自動設計システム[10]を改良し続けて，ソフトウエアグループのみならずシステムグループを含めて経営指標上での改善を推進する責任部署になる. ここに説明した設計方式は各種に標準化されて自動化向きの配慮なされている.

大目標は労働集約的作業から知中心の作業に進化することにある.

組込システムは一種のハードウエアである. コンピュータの開発では, 正常な動作を保証する為に膨大な計算機時間が費やされる. 組込システムは一種のハードウエアであるなら, 全ての可能なケースについてコンピュータで試験されるべきだ. 通常のシステムでは不可能であっても, 本構成方式では可能な組合せのケース数が大幅に少なくなっており, 可能な筈である.

他の戦略的な方法は「高加速高効率ソフトウエアテスト法, (Highly Accelerated and Yield Software Testing, HAYST)」を使うことである. この方式は「実験計画法」に似ている. すなわち, テストケースの選定に「直交表」を使う. (訳注 : 実験計画法は, 「直交表」で絞った組合せの条件で「実験」をするのだが, ここではテストをする.) 通常のテスト作業では, 負の指数減衰状に飽和してしまう. しかし, この方式は計画した到達点迄直線状の伸びになる. 従ってシステムテスト段階では大変有用なテスト方式である.

ここに説明した方式は, オブジェクト指向設計 (OOD) にも適用できると筆者等は信じている[26]. 現在, OOD が広く採用されている. OOD には各種の能力があるが, 中核になるのは, 対象を把握することである. 即ち, ソフトウエア設計者は対象の振舞いを理解し, それをFSMモデルで実現する. 殆どの人はこの概念に不慣れであり, 数多い変形を作りだす. しかし, 中核が掌握できれば, 単純な階層構成 FSM で充分に構成できる.

これにはオペレーティングシステムの進化を要する. FSMモデルは対象を掌握する自然な方式なのであるから, どんなOS であっても, 旧来の単純なスタートストップの延長として完全なイベントドリブンにすることが最も自然である. するべきことはごく単純であり, 新システムは旧来機能をそのまま含める.

ソフトウエアの世界は余りにも保守的に過ぎる. 今や世界を労働集約形態からきたるべき知の世界に向けて知を作り出し集積するように変わらねばならない.

6. まとめ

本論文は, 組込システムの究極的構築法を報告した.

1. 組込システムは, ハードウエアの延長であり, 所謂 IT 系ソフトウエアとは違う.
2. このソフトウエアは階層構成のFSM群として構築する. その背景,「モノ」と「プロセス」の両側面, イベントドリブン OS と図式的手法を強化した開発成果を報告した.
3. 約 3 状態の単位的FSMとこのFSMを単位とする階層構成であり, これら両者は最簡単な方式である. 従ってこのプロセスは究極的なものと言い得よ

う.

4. これは組込システムの開発から出発したが, これはOODでも可能である. システム設計とプログラム設計の両者ともに，将来は自動設計にできる. この視点からも，これは究極的な設計方式である.

次の課題はこれを産業界で大規模化して行き, 参画する人々にその利点を享受して頂くことである.

謝辞

著者のひとり河野は日立製作所の経営者から従業員各位に研究/開発/フイードバックを重ねる機会を与えられたことを深く感謝する. 著者等は埼玉大学情報システム工学科で「組込システムの開発」に加わられた学生諸君に感謝する. 著者等はソフトウエアクリエーションプロジェクトに参画された方々の研究成果に感謝する. また著者等はダニエル・ホーガン氏の注意深いチェックと入念な訂正に感謝する.

参考文献

[1] Z. Koono, H. Chen, H. and B.H. Far: Software Systems for Embedded System Business, *IPSJ technical report*, SIG SE 139-4, 2002.10. (in Japanese)

[2] Z. Koono: Processor Systems in High Integration Age, *Joint Conference of Four Electrical Institutes 1979*, No. 27-3, 1979. (in Japanese)

[3] A. E. Joel: Bell System Features and Services, *International Switching Symposium '79*, 1979.

[4] Z. Koono, T. Kondo, M. Igari and M. Soga: Structural Way of Thinking as Applied to Good Design (Part 1. Software size), *IEEE COMSOC Global Telecommunications Conference 1991*, pp.24.3.1-8, 1991.

[5] Z. Koono, H. Chen and H. Abolhassani: An Introduction to the Quantitative, Rational and Scientific Process of Software Development (Part 1), *Software Methodologies, Tools and Techniques 2007*, pp.361-371, H. Fujita and D. Pisanelli (Eds) New Trends in Software Methodologies, Tools and techniques, IOS Press, 2007.

[6] Z. Koono, H. Chen, H. Takano and S. Morimoto: Ten Years Education of Embedded System Developments by Student Teams, *26th Software Quality Symposium* pp. 171-174, JUSE, 2007. 9. (in Japanese)

[7] Z. Koono, H. Chen and B.H. Far: Expert's Knowledge Structure Explains Software Engineering, *Joint Conference on Knowledge-Based Software Engineering (1996)*, 193-197.

[8] H. Chen, B.H. Far and Z. Koono: A Systematic Construction Method of an Expert System Used for Automatic Software Design, *Journal of Japan Society of Artificial Intelligence*, Vol. 12, No. 4, pp.616-626, July, 1997.

[9] Z. Koono, H. Chen and H. Abolhassani: A New Way of Automatic Design of Software (Simulating Human Intentional Activity), *New Trends in Software Methodologies, Tools and Techniques*, Fujita, H.

and Mejiri, M., (eds.), p. 361-371, IOS press, 2006.
[10] Z. Koono, K. Ashihara and M. Soga: Structural Way of Thinking as Applied to Development, *IEEE/ IEICE Global Telecommunications Conf. (1987)*, 26. 6. 1-6.
[11] Z. Koono and H. Chen: Structure of human Design Knowledge and The Quantitative Evaluation (Part 1/ 2), *Technical Report of IEICE* KBSE2003-57, pp. 67-72, 2004. (in Japanese)
[12] G.J. Myers: Reliable Software Through Composite Design, Petrocelli/Charter 1975.
[13] H. E. Vaughan: Research Model for Time-Separation Integrated Communication, *B. S. T. J.* Vol. 138, pp. 909-932, July 1959.
[14] CCITT: Specification and Description Language (SDL), *Recommendation* Z.100. (1976).
[15] Z. Koono and B.H. Far: High Quality Design Using SDL Technology, *SDL Forum '95 with MSC in CASE*, Braek, S. and Sarma, A., (eds.), p. 139-150, Elrsvier Science 1995.
[16] IBM: HIPO-Design Aids and Documentation Technique, CG20-1851-1, IBM 1975.
[17] Z. Koono, H. Chen and H. Abolhassani: An Introduction to the Quantitative, Rational and Scientific Process of Software Development (Part 1), *Software Methodologies, Tools and Techniques 2007*, pp.361-371, H. Fujita and D. Pisanelli (Eds) New Trends in Software Methodologies, Tools and techniques, IOS Press, 2007.
[18] Z. Koono, H. Chen and H. Abolhassani: An Introduction to the Quantitative, Rational and Scientific Process of Software Development (Part 2), *Software Methodologies, Tools and Techniques 2007*, pp.372-390, H. Fujita and D. Pisanelli (Eds) New Trends in Software Methodologies, Tools and techniques, IOS Press, 2007.
[19] K. Hiyama, N. Mizuhara, K. Mochizuki and Z. Koono: A software system for electronic switching system using distributed state transition method, *IEEE COMSOC ICC'82*, pp. 5G.3.1-5, 1982.
[20] K. Hiyama and Z. Koono: Uniform software construct for digital switching system, *Hitachi Review,* Vol. 31. No. 5., pp. 263-268, Oct. 1982.
[21] K. Mizuno M. Kusama, M.W. Medin and W.C. Garraty: DX series of wide application digital communication controller, *Hitachi Review,* Vol. 31. No. 5., pp. 275-280, Oct. 1982.
[22] T. Ohtsubo and T. Aizawa: Fully-digital switching system HDX-10, *Hitachi Review,* Vol. 31. No. 5., pp. 269-274, Oct. 1982.
[23] Z. Koono, T. Kimura, M. Iwamoto and M. Soga: A Stored Program Controlled Environmental Function Tester Based on FMM/SDL Design, *International Switching Symposium 1987*, pp. B. 9. 5. 1-7, 1987.
[24] Z. Koono, S. Matsumoto, R. Ozaki, M. Soga and K. Ozaki K: An Environmental Simulation Tester as Applied to Traffic Characteristics Evaluation, *Proc. of The Twelfth International Tele-traffic Congress*, pp. 1284-1290, 1988.
[25] M. Yoshizawa, K. Akiyama and T. Sengoku: An introduction to Highly Accelerated and Yield Software Testing, 2007, JUSE Press. (in Japanese)
[26] T. Wang and Z. Koono: Using Extended Finite State Machine Model for Object Oriented Design, *Technical report of IPSJ*, SE-107-16, pp. 121-128, 1996. (in Japanese)
[27] B.W. Boehm: Software Engineering Economics, Prentice Hall, 1981.
[28] Software Engineering Center: White Paper of Software Development Data, Software Engineering Center IPA 2005, 2006, 2007.