ソフトウェア工学 107－15
（1996．1．19）

拡張有限状態機械をモデルとするシステムの設計方法

河野善彌，ベルーズ．H．ファー
埼玉大学　工学部　情報システム工学科
koono@cit.ics.saitama-u.ac.jp

この報告は，組込みシステム等を高品質に設計する方法を述べている．高品質に設計するには，系統的な設計方式がいる．本方式は有限状態機械でモデル化するシステムを，階層的に展開して拡張有限状態機械群により構成する．上位の設計では状態を持つ系でも機能と同様に扱い，拡張したデータフローを分割し，階層的に展開し，階層的な拡張有限状態機械の群を求める．以後メッセージシーケンスチャートから状態遷移図，最後に構造的なプログラムの完成までの一連の手続きを説明している．高品質を達成する為に，段階毎に図面を残し入念にチェックする．図面とチェックポイントについても述べている．本方式を使うと熟練者ならならば1件／KI程度の品質が達成できる見込みが立つ．オブジェクト指向設計にも参考になる．

和文キーワード　組込みシステム，設計方式，有限状態機械，SDL，オブジェクト指向，ドキュメント

A Design Method For Systems By Modeling Of Extended Finite State Machine

Zenya KOONO and Behrouz H. FAR

Department of Information and Computer Sciences,
Faculty of Engineering, Saitama University
koono@cit.ics.saitama-u.ac.jp

This paper reports on a high quality design method for embedded systems by modeling of Finite State Machine (FSM). A high quality design requires a systematic design procedure. This method details a system, a FSM, and then it is decomposed to Extended Finite State Machines (EFSM's). At the high level design, the FSM is decomposed by data flow dividing, it is partitioned to hierarchical EFSM's. Using Message Sequence Chart, state transition diagrams are derived, then they are systematically detailed leading to final source codes. In order to achieve a high quality design, design steps are recorded in small steps, and checked carefully and rigorously. It is expected that ppm order quality may be achieved. The idea in this paper is found useful useful for Object Oriented Design.

Keywords Embedded system, Design method, Finite State Machine, SDL, Object Orientation, Design documents

1．　はじめに

この論文は，組込みシステム等状態を持つ系を高い品質で設計する系統的方法を説明したものである．また，同時に発表する論文[王９５]と共にオブジェクト指向設計の参考になる事も意図している．

組込みシステムは，一般のソフトウエアより厳しい品質条件が課される．例えば，自動車の運転系に組込まれたソフトウエアの誤りは，重大な人身事故を引き起こしかねない．高い品質要求を満たす為には通常のソフトウエアとは異なる技術が発達する．事務計算等のソフトウエアは状態構造を持たない．組込みシステムはこれらと異なり，

＊状態構造をもつから，その為の制御構造，

＊状態構造を持たない一般と同じプログラム

とから成立ち，通常のソフトウエアのスーパーセットと看做せる．

本論文は，状態構造を持つ電話交換システムの仕様記述用に開発された SDL (ITU - System Description Language) 分野での経験と技術を整理したものである．SDLは一般の系を抽象化した有限状態機械 (Finite State Machine, FSM)，相互に通信し全体として動作する拡張有限状態機械 (Extended Finite State Machine, EFSM) をモデルとして用いる．

現在オブジェクト指向設計がクローズアップされている．これはある程度FSMモデルを取り込んでいる．SDL Forum '95ではオブジェクト指向設計とSDLの関係が幾つもの論文で論じられた．この論文はSDL '95での発表論文[Koono95]を補足し，更にオブジェクト指向設計に参考になる事を追加した．

2．　基本原理

2．1　高品質を達成する方法の原理

昔はバグ率が小さいソフトが高品質ソフトであった．現在では，品質を構成する副特性群に展開し，これらで品質を規定する．このように拡張された高品質を達成するには，

＊機能　　＊品質上の諸要求条件

の両者を正確に展開して設計する事が必要である．この論文では両者を「拡張仕様」と考え，これらを正確に実現する方法を検討する．

ソフトウエアの設計[河野９３]は，他の設計や人の作業[林８４]と同様に，人の知的処理の連鎖と看做せる．人の知的作業はある確率で誤り，ソースコードまで伝搬し誤りになる．全体の誤り率は，

＊人の知的作業の誤り率／単位知的作業

＊対象設計作業規模あたりの単位知的作業数

とで評価できる．はじめに，

＊誤り率／対象設計作業規模 = E ..誤り作込み率

を基として高品質すなわち低誤り率の為の作業条件を求めた結果[河野９３]を再掲する．

誤り作込率Ｅは，初心者のＥ＝１００件／K1から熟練者の１０件／Ｋ１の範囲にある．この品質では実用に供し難いから，チェックやテストを行う等，色々な運用方法をとる．諸方式を定量評価すると，

１．設計したのみで何らの誤り摘出をしない時，

＊ソフトの残留誤り率＝E

２．設計に続くチェック（テストではその設計作業）も設計と同様に誤る．設計とチェックの両者が共に誤る時に，結果は誤りになる．

＊残留誤り率 R = E × E', ('はチェックの場合)

簡単の為に設計でもチェックでも，誤りはランダムであり，その率は等しいと仮定すると，

＊残留誤り率 $R = E^2$

このようにチェックすれば残留誤り率が大幅に減る．Ｃ回のチェックすれば，

＊残留誤り率 $R = E^{C+1}$,

となり，更に大幅に減る．

３．設計作業を知的作業数の等しいM区間の縦列接続で構成し，区間毎に１回チェックをすれば

＊残留誤り率 $= E^2/M$

と１／Mに低下する．

Ｅ＝１００件／K1の初心者の場合には，

１．設計後チェック無しなら　Ｒ＝１００件／K1

２．１回チェックすれば　Ｒ＝１０件／K1

３．M= １０，１回チェックなら１．０件／K1

となる．そこで，

＊設計を小さい区間に分けて，

＊設計結果を記録し，

＊入念に厳密なチェックを繰り返す，

事により，残留誤り率Rは幾らでも減らせる．企業組織としてppmの桁の残留誤り率をTQCで達成した実例では，組込みシステムの開発に１０種を上回るドキュメントを用いている．[倉原９０]

このように高品質を達成する為には，作業工程が

系統的かつ階層的な構成で，作業の小さな進行毎に結果を記録できる方式が要る．

２．２　拡張有限状態機械をモデルとする方式

ソフトウエアの設計の「モデル化」とは，「モデルを基本形式の規範として作業する事」とする．有限状態機械モデル（ＦＳＭ，Finite State Machine）は，一般の機械等を抽象化したモデルであり，組込みシステムのモデルとして最もふさわしい．

状態を意識して設計する時，複雑度が高いと状態数が爆発的に増えるので適当でない，との声がある．この問題点は次のように避けられる．

あるＦＳＭの動作を，相互に通信する複数のＥＦＳＭ (Extended FSM, EFSM) で構成して有機的に一体と等価な動作をさせる．

すなわち，状態数ＭとＮの二つのＥＦＳＭに相互に連係動作させ，系として状態数Ｍ×ＮのＦＳＭと同じ動作を行わせる．

方式	状態数	プログラム規模
単一ＦＳＭ	：ＭＮ	ＭＮｋ
複数ＥＦＳＭ	：Ｍ＋Ｎ	（Ｍ＋Ｎ）ｋ

Ｍ，Ｎが相互に独立な１０状態の二つのＥＦＳＭでＭＮ＝１００状態のFSMを実現すれば，複雑度は１／５に減る．状態遷移プログラムの量は，状態数の一定率（ｋ）であるから，同率で減る．（実際の例は[河野９１a] に示してある．）

FSMは，現実を正確に捕えるよいモデルなので，系統的な設計に適している．そこで，

＊系をEFSMの集合体に系統的に展開する方法，

＊(E)FSMを系統的に設計する方法，

＊設計履歴を詳細に示すドキュメント，

の三者があれば，高品質な設計が実現できる．FSMをモデルとすると，系統的な設計ができる事は，[野口９０]等で良く知られている．

複数のＥＦＳＭで単一のFSMと等価な動作を行わせるには条件がある．図１[Koono87]の上部は，全体としてあるEFSMに代る為に各ＥＦＳＭが順次状態遷移を実行する様子を示し，下部は制御構造を示す．単一ＦＳＭの為の一連の動作を継続して行わせるので，各ＥＦＳＭ間の通信キューを外部入力より優先させる．状態遷移先の決定は後の図７に示す．

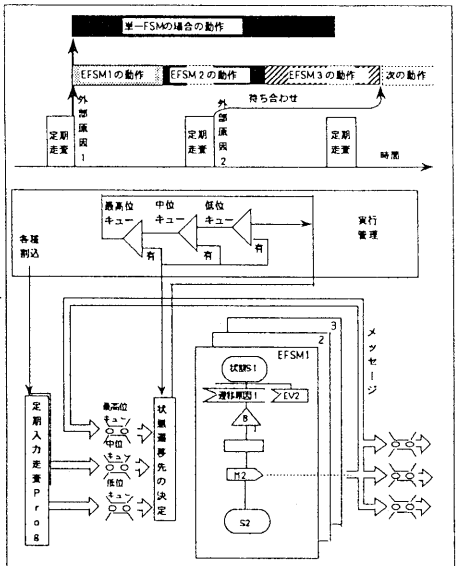

図１　EFSMの代替動作

諸概念を以下のように定義する．

＊状態：ある視点から見て継続的な状態にある事

＊遷移原因（イベント）：状態遷移を起こす事象，一般に関連する情報を伴う．

＊状態遷移：遷移原因により起動され，終結時の状態に至る遷移のルート．(JISの規定と異なり)分岐して複数終結状態になる事も許す．

＊誤りを防止し制御を簡単化する為，状態遷移実行中は遅延を許さない．タイマーEFSMを設けてタイミング指令を与え，タイミング経過報告を受ける．

＊メッセージ：　遷移原因と関連情報を指定されたEFSMに伝達する手段．

３．　設計方法とドキュメント

３．１　考え方

本方式では，システム全体をひとつのFSMと看做して相互に独立な構成要素（ＥＦＳＭ）に展開し，詳細に具体化する．上位の設計は構成方式の決定であり，システムのアーキテクチャを支配し，その良否は製品の機能，ソフトウエア開発規模，長期間の機能変更や拡張の容易性等を決める．良い構成方式を導く為の系統的な方法を３．１で説明する．設計結果には他の方式同様に設計者の能力等に依存する所があり，たとえ多様な候補を考えた上でも，製品の事業計画から技術面，市場状況，使用状況や使用

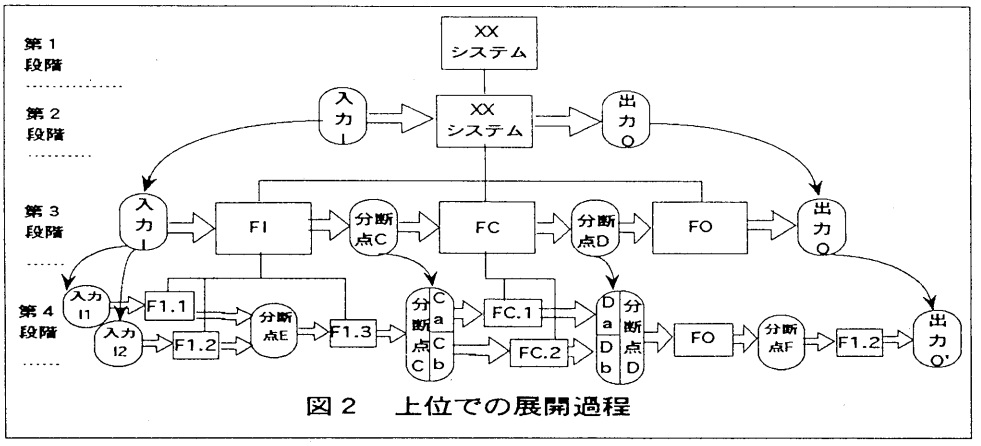

図２　上位での展開過程

者心理，実現技術に至るまでの考慮を織り込んで，選択し決定するにも高い能力が求められる．

誤り少ない良い設計をするには，極力自然言語概念の世界で展開し，一番最後で一挙にコードに変換する方が良い，とJackson, Myers等多くの人が指摘している．これは誤りの発見の容易性からも当然である．設計者が諸概念を論理的に思考する過程を正確に行う事が鍵になる．技術以外に自然言語レベルで豊富な語彙や正確で豊かな表現力等が重要である．始めに示したように，誤り少なく設計し，厳密にチェックする事が不可欠で，その正しさが全体に影響するのは前述の諸式からも明らかである．

設計方式としては，「多くの人に，より容易に出来るようにする」事までしか出来ない．ここまでの前提や条件を理解して使用する事が必要である．

３．１　システムレベルの展開

システムを相互に独立な構成要素にするには，幾つかの方法がある．ここでは，

システムを拡張した情報の流れの分割を中心に

単機能的ＥＦＳＭ群にする方法

を説明する．図２はその作業階程である．

システムは通常複合機能なので，目的的な概念名，XXシステム，としか命名できない．これを一つの機能箱で表わす．これが第一段階である．

第二段階では，システムを概念的な情報の流れに変える為，イベント等を含めた概念的な入力と出力を与える．考えられる入出力のうち最も抽象度の高いものを選ぶ．図は単一の入出力を示すが，複数の場合も，場合により違う事もある．これらの時には別図にして作業する．原則は次のとおりである．

＊出力をまず決め，これを作り得る入力を決める．

＊上位の機能箱の機能／効用が，この段階の入力，機能概念，出力の概念的な流れで果たせる事．

第三段階では，拡張したデータフローの分割により，入出力や機能の階層的展開をする．ソフトウエア工学では機能とは入力から出力への情報変換であるが，外部機能的でも状態を持ったもので良い．

図２のように，拡張したデータフローの分断点，インタフェイス概念，を先に決める．機能を先に決めると，その名称は中心的概念でインタフェイス概念に直結しない．しかし，分断点（インタフェイス概念）を先に決めると，曖昧度が大幅に小さくなり，また中心機能概念も逸脱しなくなる．インタフェイスあるいはデータはこのような重要な性質をもつから，データをまづ第一に書き，次いで機能を，左から右に実現の為の流れを書く．（バブルチャートはデータが明示されないので不適当．）

正確性の為に次の原則を守る．

＊各インタフェイスや機能の設計（展開）は，等しいか，正確な具体化か，正確に階層的に展開する場合のみに限る．

＊メッセージも含めた入出力や分断点（インタフェイス概念）は，最上位から最下位の具体的インタフェイス概念に至るまで，機能概念の展開と合わせ正確かつ明確に展開する．

中間の分断点（インタフェイス概念）には以下の条件がある．

＊拡張したデータの流れを情報変換の連鎖と見る．
＊抽象度の高いインタフェイス概念で分断する．
　＃入出力や分断点（インタフェイス）は定義表を作り，構成，名称，定義を記す．
MyersのS－T－S法[Myers76]は，分断法の例である．入力の概念を持ちながら入力から最も遠い概念（最大抽象入力点）と，出力の概念を持ちながら出力から最も遠い概念（最大抽象出力点）とを選び，流れを分断して３区間化する．
＊一度に多数の分断をしない．（分断を繰り返す事になるが，各機能を極力単機能化する方がよい．）
＊状態を持つ系では，（自動販売機の「硬貨投入」と「投入金」のように）
　動作的な遷移原因，
　概念的で遷移原因に付随する情報，
の二つの視点がある．現段階では後者に留める．

次ぎに入出力や分断点で挟まれた区間の機能（名）を決める．システム（サブシステム）等を展開しても未だに単機能的な概念に至らない時は，目的的な概念名のサブシステム／デパートメント等にする．展開するうちに，目的と言うより機能的な概念が見え始めから，以後は極力単機能的概念の名称を与え，「○○を□□する」と表現する．

状態を持つシステムや機能部分では，（Myersの言う）Source－Transform－Sink の３機能部分のうち，Source と Sink が同一実体になる場合がある．（図２第４段階のＦ１．２）これらは後で一体化する．

入出力や分断点（インタフェイス）と機能で構成された鎖状データフローは，対応する上位の入力，機能，出力の流れと正確に等しくする．
＊展開後の入出力／データ／インタフェイスは上位と等しいか，正確に具体化されているか，階層的に具体化されている事．
＊下位機能は上位機能の部分概念である事．
＊入力，最初の機能，インタフェイス，次の機能と流れを順次読み上げてみて，違和感がなく，その流れにより上位機能が果たされる事が理解できる事．

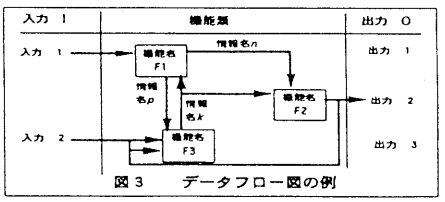

図３　データフロー図の例

幾つかの入出力がある時や場合により機能が違う場合には，それぞれ別のデータフロー図に展開する．（共通な機能の名称は統一する．）各場合のデータフロー図を纏めた図３のデータフロー関連図を別途に作り，全体のバランスを調整する．

上位の機能は，下位の何れかの機能を欠いては成立せず，相互に独立な構成成分に分割された．また，単機能概念，極力小さな設計の進行毎，区分毎の分割だから，単機能的に分割されている．

この展開を繰り返して第４段階以降を得る．何時までも全体を１枚の図で見渡す事はできない．適当な段階からは，あるインタフェイスで挟まれた機能部分を取出し，別図で展開する．

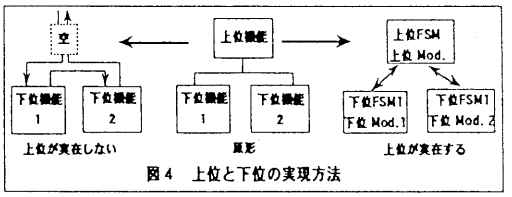

図４　上位と下位の実現方法

上位機能と下位の鎖状データフローが等価な時，
＊上位機能を空とする方法（e.g. 構造化分析），
＊上位機能に実体を設けて，下位とプログラム的な結合関係を持たせる方法，
の２種がある．図４はこれを示す．極力単純でもよいから上位機能に下位をコールする機能，または分配的なＥＦＳＭにする等してプログラム実体を与え，下位機能の独立単機能化を図る．（上位を空にすると，下位のプログラムは，自分自体の機能のみならず，上位で持つべき各種統合の機能をも併せ持ち，実体が単機能でなくなり，独立性を失う．
（構造化分析での空の展開やSDLのブロック概念はなるべく使わない方が良い．）

ある（Ｅ）ＦＳＭを展開して，２～３階層，全体の（Ｅ）ＦＳＭ数が５～６個位で一応止める．必要ならこれらの何れか一つを取り出して，同じ作業を続ける事を繰り返す．これまで，状態を持つか否かにかかわらず機能として扱ってきた．展開を続けると，状態を持つ１機能を状態毎の下位機能の群に展開する危険性がある．展開の度に，
＊状態を持つか，
＊入出力関係が構造的な内部記憶を持つか，
を確認して，状態毎に展開する事を避け，限度にきたら次の３．２の段階に移る．

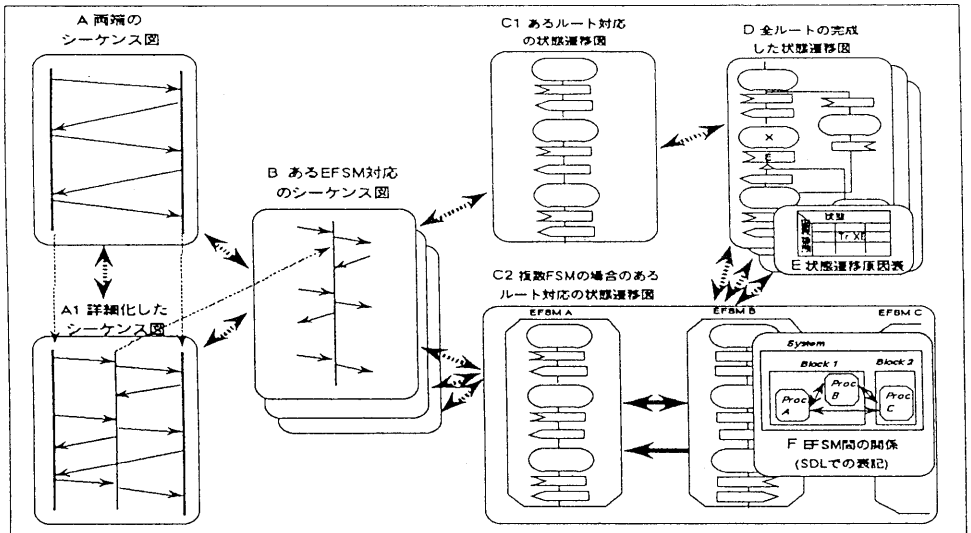

図５　　各EFSMの状態遷移の設計手順

３．２　各EFSMの状態遷移の設計

系全体の外部的な振舞から，数個単位の各EFSM毎の状態遷移図迄の設計手順を図５に示す．

図Ａの（メッセージ）シーケンス図を出発点に用いる．矢線の先は到達先，根元は送出元を示す．これは入出力を明記したデータフロー図を時間を考慮し，縦線（あるＥＦＳＭに対応する）で折り返したものにあたる．従来同様に，次の条件で作業する．

＊各矢線は既に定めた外界や各ＥＦＳＭ間のインタフェイス概念に等しいか，階層的展開等の正しい具体化である事．

＊矢線に名称を添える．入と出により果たされる機能は，そのＥＦＳＭの機能定義（概念）の正しい具体化に限る．

図Ａは着目対象と外界（またはあるＥＦＳＭ）との最も単純なシーケンス図である．次にこれを図Ａの縦線１本で代表させた(E)FSM群の為の図A1のシーケンス図に変換（詳細化）する．図Ａから図Ａ１に忠実かつ正確に変換する．この時の増加は多くても３ないし５程度の増加にとどめる．多すぎると著しく誤りが多くなる。この展開を最上位から始めて最詳細レベルに至るまで展開を繰り返し，正確に具体化し，段階毎にチェックを繰り返す．

シーケンス図はある場合の操作例や手順例を示すに過ぎない．少なくとも正常動作，できれば代表的な数種の場合を準備し，それぞれ具体化する．並行して，各ＥＦＳＭ間のインタフェイスのメッセージ定義表や一覧表を作る．メッセージ定義表は，遷移原因とその付随情報について，構成と定義を示し，作業が進むにつれて段階的に詳細化し，最後にはニーモニックコードに至る迄記載する．データフロー段階での概念的なデータ定義は多くの場合に付随情報に引き継がれる．

図Ｂはシーケンス図中の１本の縦線で，何れかのＥＦＳＭに当たる．これを図のＣ１，Ｃ２等の着目する動作ルートについての状態遷移図に変換する．状態はシーケンス図上で決定でき，入射矢線は遷移原因に，出射矢線は遷移中の他ＥＦＳＭ宛のメッセージや出力になる．複数の図で上部シーケンスが共通なら，遷移ルート中の決定で上部を併合する．

全状態の遷移原因毎の遷移ルートに識別名を与える．図Eの状態遷移原因表を準備して対応交点欄に（JISのように終着状態でなく）遷移ルートの識別名を記す．記載後の表の空欄は状態遷移のうち考慮に漏れたものを示すから，脱落した状態遷移ルートを補充する．（漏れ防止に極めて効果的である．）

通常ではないが起こりうる異常（準正常）動作は全体としての設計方針（例：正常以外は全て無視する，最寄り状態から再試行する等）やその状況の特殊性を考慮して決める．対人インタフェイスでは，

1正常遷移ルートに対して3～5の異常（準正常）対応の遷移ルートがあり，その設計工数は正常な場合の数倍を要する．（この徹底さが，通常のソフトウエアと大きく異なる．要注意点．）

全て作業は，段階毎に繰り返し厳密にチェックする．その他にシミュレーションチェックがある．古典的なチェック方法は，机上テストする場合を決めておき，最初の外部入力から始め全状態遷移図群にまたがり動作を逐一入念にシミュレーションチェックした．これは莫大な工数を要するにもかかわらず，誤り摘出漏れが多く不適当である．容易に追える高々3～5ＥＦＳＭへの展開，それぞれの状態数が１０以下毎にチェックする．正しく作業し，チェックしてあれば，部分シミュレーションチェックする方が効率も正確さも遥かに良い．

3．3　プログラムレベルの設計

図6に示す図Aの原理的な状態遷移図から具体的なプログラム迄の設計手順を説明する．

この図は決定（分岐）とメッセージや出力を含むが、状態遷移の為の「機能」が無い．これを図Bの左の各状態の正確な定義図から求める．各EFSMを単機能的にすれば，1状態に関係する資源は少ない．図Gの資源毎の状態遷移の前後の表の交点は状態遷移の為の「機能」にあたる．これらは予め詳細化し確認しておき、表から図Cのように挿入し，状態遷移のルートを状態遷移プログラムにする。

メッセージや出力も，種別毎に予め原形プログラム群を作り，単体動作を確認しておく．（これらは，各EFSMを単機能的にすると標準化でき単純になる．）使用時にパラメータを指定してメッセージや出力のシンボル対応に置換する．

図Cのように状態遷移は複数の遷移後状態を持つ事が多い．これでは複数出口を持つ悪いプログラムになる．各終着状態の直前に「次状態定義」を挿入し，プログラムを単一入口／出口に変換する（図D）．仕様記述言語SDLでの詳細化は図Cで終わる．しかし，このように構造的なプログラムに作り替えてしまうと，以後は通常の構造的プログラムとして扱え，設計情報を後段に接続して構造化プログラムレベルのCASEツールを使用できる．

4．　検討

4．1　状態遷移の制御

先に省略した図1の状態遷移を説明する．各状態遷移プログラムは図7の状態遷移原因表の交点に対応させてある．表の各欄は最後には対応状態遷移プログラム（名）で，図1の「状態遷移先の決定」内にある．各状態遷移の実行の最後に次状態定義により各EFSMの状態記憶を更新する．キューから取り出したメッセージは宛先EFSM情報を持つ．指定EFSMの現在状態とメッセージ中の遷移原因とで状態遷移原因表を引き，該当する状態遷移プログラムの実行にはいる．そこで図6Dに破線で示した当初

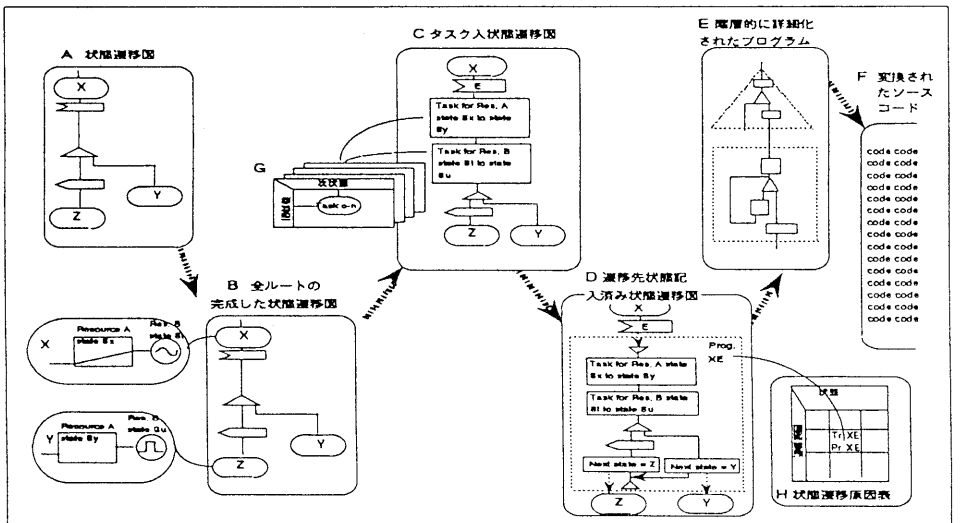

図6　プログラムレベルの設計手順

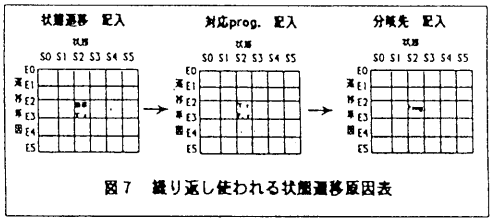

図７　繰り返し使われる状態遷移原因表

意図どおりの状態遷移を行う．

４．３　本方式の評価

１９８０年以来各種の機会に改善しながら方法を改善してきた．一番最近の教育での例を報告する．

大学の２年生に各種ドキュメント，構造化プログラムとCASEツールを教育し，３年生進級後に設計方法を説明し体験的な演習として数名単位のチームに各種の自動販売機を設計しテストさせた．規模は数百行で，要領よいチームで１００人時間～熱心に議論しレビューしたチームは３００人時間を費した．現在，学生実験の一テーマとして他チームの品質保証作業の体験としてテスト後の残留誤を摘出する．

最終段階の開発者によるテストでの誤り摘出率は０～１８件／KIの範囲で平均約９件／KIであった．その後の学生の品質保証段階でも若干の本質的な不良が発見されているが，上記テスト後の残留誤りは平均して上記の数分の１程度の見込みである．初心者は１００件／KIの不良を作込み，大半の時間をデバッグに費やす．その人達が「作って段階的にテストして軽微なバグで仕上がる」事を経験している．

本方式で熟練者が作業すると開発者システムテストの不良摘出率で１件／KI以下に留められる．入念な開発者システムテストの完了段階では残留不良率は１／１０位になり残留誤り率０．１件／KI，品質保証段階でブラックボックス的に十分にテストすれば更に１／１０程度に抑圧できる．技術を集積すれば０．００１件／KI，ppmの桁の品質達成を見込め，[倉原９０]の実績に近づける．

作業を効率化するには，かような方法のできるCASEツールが望まれる．また，誤りは工程の欠陥の反映だから，高品質達成には基本的な設計方法以外に充実したチェック技術，ノウハウの積み上げと従事者の技術レベルの高さが不可欠である．

５．結び

設計作業の小さな作業進行毎に，結果を記録し，厳密にチェックし，残留誤りを大幅に低減できる．

これは系統的な作業方式により容易になる．組込システムに適した（拡張）有限状態機械をモデルとする系統的な作業方法を報告した．これは，通信業界で１９６０年代から適用しているFSMモデルによるSDL技術の一つの発展形である．

現在クローズアップされているオブジェクト指向設計もまたＦＳＭモデルの一応用である．本方式を応用した[王９５]とともに，各位のご参考になれば幸いである．

謝辞

方式を開拓した日立製作所桧山邦夫氏とそのチーム各位，御支援下さった三浦武雄博士，失敗しても挑戦を続ける事に寛容であった上長各位，実用化に努力された各位，最後に新米教師の教育の中で努力して頂いた本学科の多くの学生諸君に深謝します．

参考文献

[王９５] 王，河野：オブジェクト指向設計への拡張有限状態機械モデルの利用，情報処理学会，ソフトウエア工学研究会資料，１９９６．１月．

[倉原９０] 倉原，内丸，岡本：技術集団のTQC，日科技連，１９９０．

[Koono87] Koono,Z., Kimura, T., Iwamoto, M. and Soga, M.,: A stored program controlled environmental function tester based on FMM/SDL design, International Switching Symposium '87, 1987.

[Koono 91a] Koono, Z.,: Structural way of thinking as applied to good design, (Part 1, Software size), IEEE Global Telecommunications Conference 1991.

[河野９１b] 河野：ハード制御・通信に用いる埋め込みシステムの設計手法，電子情報通信学会，技術報告ＳＳ９１－２７，１９９１．

[河野93] 河野，大坪：ソフトウエアの誤りと除去の評価，情報処理学会，ソフトウエア工学研究会ＳＥ９５－５，１９９３．

[Koono95] Koono, Z. and Far, B. H.; High quality design using SDL technology, Seventh SDL Forum, 139-50, 1995.

[野口９０] 野口：ソフトウエアの論理的設計法，共立，１９９０．

[林８４] 林喜男：人間信頼性工学，海文堂，１９８４．

[Myers78] Myers, G. J.: Composite structured design, Litton Educational Publishing, 1978. 国友，伊藤訳：ソフトウエアの複合／構造化設計，近代科学，１９７９．