電力系統制御用ソフトウェアの階層化分割法について*

——階層化分割法の定性的検討——

奈良宏一**
辻　俊彦***

(昭和52年9月30日受理)

Hierarchical Decomposition of Software System for Power Systems Computer Control

— Qualitative Studies of Hierarchical Decomposition —

by Koichi NARA and Toshihiko TUJI

The software system of computer control for power system must be designed so as to satisfy some specific requirements. Two of the important requirements are increasability and the maintenability of the software system due to the increase of the power systems' apparatus. Some of other requirements are the multipurpose useages of large data from wide service area, and that the power systems' control be highly reliable. The software structure must be constructed comprehensible for engineers of design and maintenance who are usually specialists for power systems' control, but not for computer programing.

So, we developed a computer software designing technique which enabled easy maintenance and is highy-reliable. The technique has been put to practical use in some power systems' control fields.

In this paper, we argue the necessity of hierarchical decomposition of the software system for the power systems' control, and propose a method of hierarchical software decomposition which realizes it easy to document, to coodinate, to maintain the software system and to pursue high-reliability, by qualitative studies of the concept of hierarchy and program models.

1. まえがき

近年，電子計算機の導入台数の増加に伴い，そのソフトウェアの製作，変更，保守に要する費用が人件費の高騰と相俟って莫大なものとなってきており，“ソフトウェアの危機”という言葉と共に，その対策に数多くの検討，提案がなされてきた。ところが，電子計算機の応用分

* 昭和51年度電気関係学会関西支部連合大会（昭和51年11月）で一部発表

** 北見工業大学電気工学科
*** 三菱電機株式会社電力系統システム部

野，プログラム設計・製作者の能力等によってその提案が画一的に適用可能とは限らない。

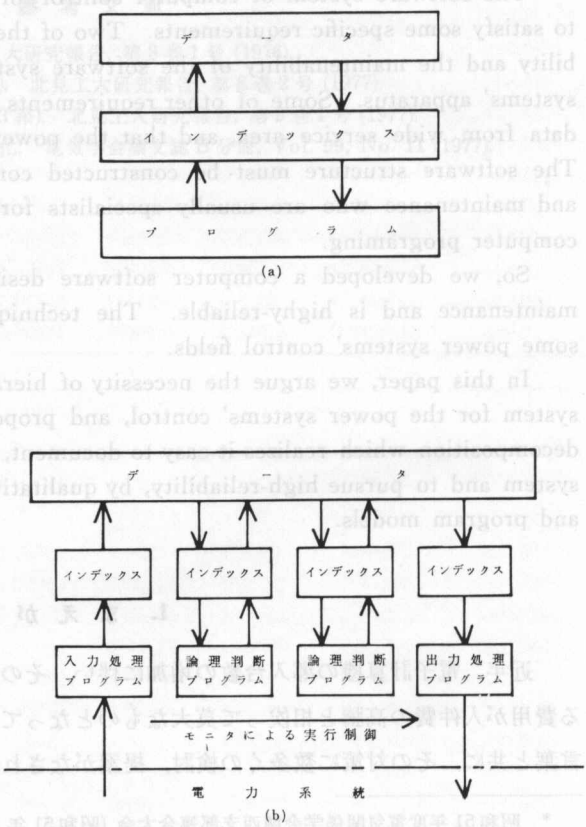
特に電力系統制御用計算機においては，電力系統の拡張に伴うソフトウェアの増設・変更，広範囲な地域からの大量データの多目的処理，制御の高信頼性といった問題があり，ソフトウェアの設計段階においてこれらの要求を充分に満足した設計がなされねばならず，また，一般に，ソフトウェアを設計・保守する技術者は電力系統制御を専門とする技術者であり，プログラム作成に充分な経験を積んでいない事から，理解し易いプログラム構造を有する事を要求される。

そのため，我々は，(1) ソフトウェアの増設・変更の容易性，(2) 高信頼性を考慮したソフトウェア設計手法（プログラムとデータの分離，入力・処理・出力の分離，プログラムのモジュール化・階層化を考慮した設計手法）を開発し，一部で実用に供してきている[2]が，本稿では，前記設計手法を拡張し，階層化の観点から，さらに理解し易い設計手法を提案する。

2. ソフトウェアの基本モデル

本稿で，ソフトウェアとは，データとプログラムを含めた全体集合を称するものとし，広義な意味での解法等は含まないものとする。プログラムとは，ある入力に適当な処理を加えて目的とする出力を得るための一連の命令集合を呼び，データとは，プログラムへの入力とプログラムからの出力を呼ぶ。

データが大量になり，1つのデータが多くの目的に使用される場合，データを一まとめにしてデータベースと呼ばれるデータの集合を形成し，プログラムとは分離して考えた方が扱い易い。データとプログラムを分離すると，プログラムがデータをアクセスする場合の指標（インデックス）が必要となり，結局，ソフトウェアはプログラム，インデックス，データという3層構造を有するモデルとして取扱う事ができる。これを図-1 (a), (b) に図示する。図-1 (a) は，ある入力データがインデックスを介してプログラムに処



図—1 ソフトウェアモデル

Fig. 1. Model of soft ware system.

理され，処理されたデータが出力データとしてデータベースへ出力される事を意味している。また，入力・出力・処理を分離して考えると，より具体的に図-1(b)のように図示できる。

データ，インデックス，プログラムは，各々，その構造を有しており，データの種類は全体のプログラムの構造に，インデックスの構造はデータベースの構造に依存して定まると考えることができる。

以上のように，ソフウトェアシステムの大半はプログラムの構造によって定まることから，本稿では，プログラムの階層化設計手法についてのみ検討する。

3. 階層化の目的

通常，電力系統制御に計算機を導入する場合，計算機の機能は単一ではなく多機能である。一般に，多機能のソフトウェアシステムを最初からトータルアプローチ的に設計することは困難であり，また，単一プログラムによって実現しようとすることは非現実的である。そのため，通常，全体のプログラムを幾つかのタスクと呼ばれるプログラム単位に分割し，個々のタスクを実行制御するプログラム（モニタ，スーパバイザ等と呼ばれる）によって有機的に実行することによって所定の目的を達成している（図-1(b)）。したがって，何らかの分割が必要であるが，この分割を階層的に行うことによって，次のような利点が得られる。

(1) 統合の容易さ

分割したプログラムが階層的であれば，個々のプログラム間の関係が明白となり，トータルシステムとしての統合が容易になる。

(2) 記述の容易さ

全体のソフトウェアシステムを把握する場合，その記述は幾つかのサブシステムに分割して記述しなければ理解が困難である。この場合，階層的に分割された記述が一般に実用的である。

(3) 保守の容易さ

ソフトウェアが階層的に分割され，さらに個々のサブシステム間の関係が少なく設計される程，一つのサブシステムの変更による他のサブシステムへの影響が小さく，保守が容易になる。

(4) 信頼性の向上

階層的分割により，一つのサブシステムがそのサブシステム内で機能が閉じていれば，そのサブシステムのダウンによる他サブシステムへの影響が小さく，他のサブシステム内で閉じている機能については，機能の続行が可能である。また，階層的分割によるサブシステムの機能の明確化，単一化，ならびに単一プログラムの小型化により，設計・製作時におけるバグ数の減少が期待できる。

4. 階層の概念

階層の基本的形態は Mesarovic 氏によれば次の3つの考え方に分割できる[3),4)]。

(1) 記述あるいは抽象におけるレベル（モデル化によるレベル＝Strata）

(2) 意志決定の複雑さにおけるレベル（制御レベルによる＝Layers）

(3) 組織におけるレベル（各々の行動の制限によるレベル＝Echelons）

これら3つのレベルに関する概念は，考え方において違った目的で導入されている。すなわち，Strata の概念はモデル化のため，Layers の概念は意志決定問題をいくつかのサブ問題に分解するため，Echelons の概念はシステムを構成する意志決定ユニット間の相互の関係を明確にするために導入されている。

これら3つのレベルの関係は，階層システムを構成する時，次の順序に行うのが合理的であることから明らかになる。

(1) トータルシステムの観点から全体の階層的記述を行う。(Strata 化)

(2) システム全体の仕事を意志決定の複雑さにより階層化する。(Layers 化)

(3) その後に組織化階層を構成し，関係を明らかにする。(Echelons 化)

ソフトウェアシステムにおいても，階層を意識するしないにかかわらず，従来から無意識にこの手順に従って設計がなされてきたと言えるかもしれない。以下に検討する手法は，設計の頭初から，この3つのレベルを意識して設計する手法である。

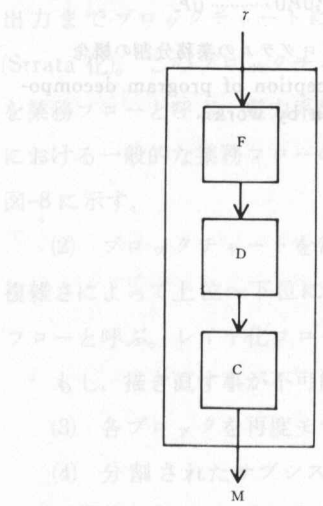
また，これら3つの概念は，プログラム実行時におけるプログラムレベルとは異質の概念であることも明らかである。

5. プログラムモデルと階層化

今，プログラムを図-2のように抽象化することは一般的である。すなわち，プログラム P は入力データ $\boldsymbol{\gamma}$ に変換 $\boldsymbol{F}$ を加え，意志決定 $\boldsymbol{D}$ を行った後，結果に変換 $\boldsymbol{C}$ を加えて出力データ $\boldsymbol{M}$ を出力する事を意味している。このモデルは2つの方法で分割可能である。第1の分割は，機能による分割であり，入力 $\boldsymbol{\gamma}=(\gamma_1, \gamma_2, \gamma, \cdots \gamma_n)$ か又は出力 $\boldsymbol{M}=(m_1, m_2, m_3, \cdots m_n)$ によって，図-3のように分割する方法である。z_i, w_i は各々分割によって派生するインタフェイス入力，インタフェイス出力である。この分割は，多種類の入力及び出力が混在し，その使用目的が相互に関連している電力系統制御システムにおいては，インタフェイス入出力 z_i, w_i が複雑に関係し合うであろう事が容易にわかる。第2の分割は，$\boldsymbol{F}$, $\boldsymbol{D}$, $\boldsymbol{C}$ も1つのプログラムであることから，図-2のモデルを $\boldsymbol{F}$, $\boldsymbol{D}$, $\boldsymbol{C}$ に適用する手法である。意志決定 $\boldsymbol{D}$ は，ある一つの決定 d_j をなす場合に，それ以前に決定されたより簡単な決定 d_i の結果 m_i を決定の判断要素として用いる場合も考えられるので，一般に 図-4 に示すように図示できる。第2の分割は，第1の分割に比べ，インタフェイス入出力がより明確になっていることがわかる。これは，入力と出

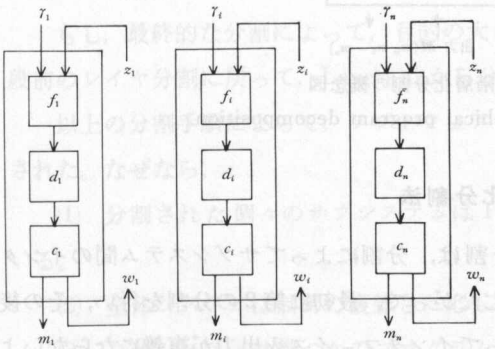
力を分離したことによる。

理解を容易にするために，上記二通りの分割をより抽象化して図示すると，第1の分割は図-5，第2の分割は図-6のように描くことができる。また，より具体的に説明するならば，第1の分割は機能（function）による分割であり，一般に分割されたサブシステムにおいて，一つの機能を閉じたループで実行することが望ましいが，入力データの処理が多目的であるような場合，機能間に複雑な関係が生ずることは避け難い。第2の分割は業務（work）による分割であり，復数の業務の連続的なつながりによって一つの機能を実現させようとするアプローチである。個々のサブシステムが他の業務とは無関係に与えられた業務のみを閉じて実行していくうちに目的の機能を達成させようという手法であり，組織を構成する場合に便利に利用されている。



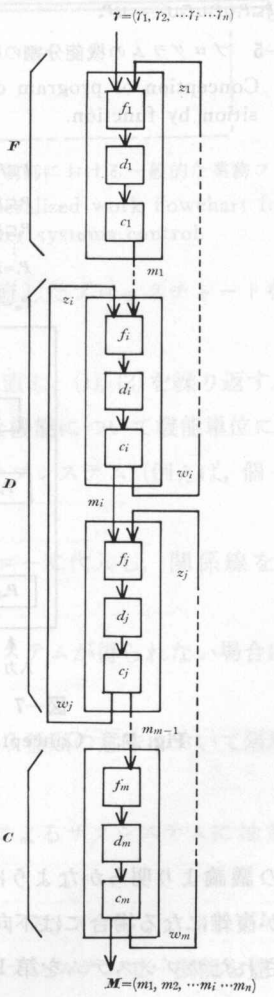
図—2　プログラムモデル

Fig. 2. Model of program.



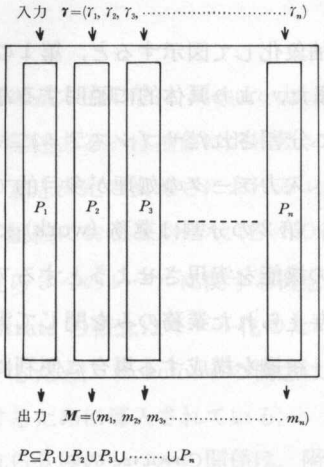
図—3　機能によるプログラムの分割

Fig. 3. Program decompostition by function.



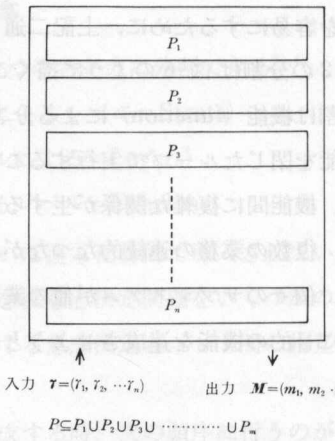
図—4　業務によるプログラムの分割

Fig. 4. Program decomposition by work.



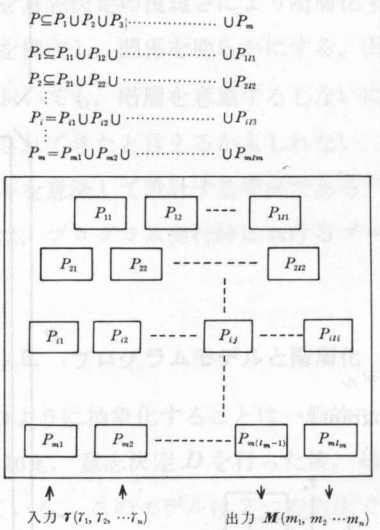
図—5 プログラムの機能分割の概念

Fig. 5. Conception of program decomposition by function.



図—6 プログラムの業務分割の概念

Fig. 6. Conception of program decomposition by work.



図—7 プログラムの階層化分割の概念図

Fig. 7. Conception of hierarchical program decomposition.

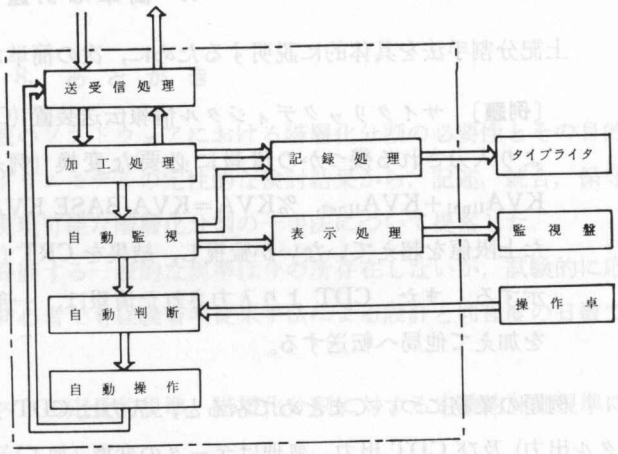
6. 階層化分割法

前述の議論より明らかなように，第1の分割は，分割によってサブシステム間のインタフェイスが複雑になる場合には不向きである。したがって，最初に第2の分割を行い，その後に，分割されたサブシステムを第1の分割によってインタフェイス入出力が複雑にならないように分割する方法が現実的である。すなわち，第2の分割によって生じたサブシステムを，サブシステム内で相互干渉の小さい部分を第1の分割によって分離する方法による。しかる後に

生じたサブシステムにさらに第2の分割を加え，再び第1の分割を行うという分割を繰り返して目的の分割を得るトップダウン式のアプローチである。より具体的には，図-6の分割によって生じたサブシステム P_i に図-5の分割を加え，図-7の P_{ij} を得る手順を繰り返すアプローチである。

上記のように階層的に分割されたソフトウェアシステムを得るには，次のような手順でソフトウェアの分割を行うとよい。

(1) ソフトウェアの全体業務(機能ではない)を大まかに入力から出力までブロックチャートに描く(Strata 化)。このブロックチャートを業務フローと呼ぶ。電力系統制御における一般的な業務フローの例を図-8に示す。



図—8　電力系統制御における一般的な業務フロー

Fig. 8. Generalized work flowchart for power systems control.

(2) ブロックチャートを制御の複雑さによって上位～下位に描き直す(Layers 化)。描き直したブロックチャートをレイヤ化フローと呼ぶ。レイヤ化フローの一例を図-10に示す。

もし，描き直す事が不可能な場合は，(1)を再度検討し直し，(1), (2)を繰り返す。

(3) 各ブロックを再度モデル化し，相互に干渉の小さな機能について機能単位に分割する。

(4) 分割されたサブシステムが目的とする大きさのサブシステム(例えば，個々のブロックが単機能となる)になるまで(2), (3)を繰り返す。

(5) 最終的に分割されたブロックを順次レイヤ化フローに代入し，関係線を整理する(Echelons 化)。

もし，最終的な分割によって，目的の大きさのサブシステムが得られない場合は，順次1段前のレイヤ分割に戻って，Layers 化から分割し直す。

以上の分割手順によって，ソフトウェアシステムは，3種類の意味において階層分割がなされた。なぜなら，

(1) 分割された個々のサブシステムは1段前の分割によるサブシステムに抽象化されている。

(2) 全体システムは制御の複雑さに応じて分割されている。

(3) 全体システムは幾つかのプログラムを結合したサブシステムで一機能を達成し，そのサブシステムの集合で全体機能を達成するように分割されているからである。

なお，上記分割によって機能面からの理解が困難とならないよう，一機能を表すように業

務単位のタスクを結合したフロー図（システムフローと呼んでいる）を作成し，機能面からの理解を助けている。[2)]

7. 簡単な例題

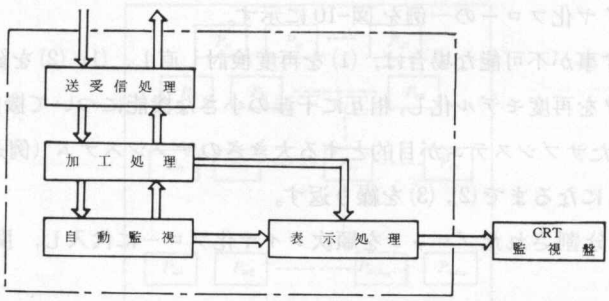
上記分割手法を具体的に説明するために，次の簡単な例題について分割を行ってみる。

〔**例題**〕 サイクリックディジタル情報伝送装置 (Cyclic Digital Telemeter=CDT) より入力される幾つかの情報に必要な変換（例えば，KVA=$\sqrt{3}$ VI, $KVA_{line}=KVA_{line1}+KVA_{line2}$, %KVA=KVA/BASE KVA）を加え，あらかじめ定められた上限値を超えていないか監視し，結果を CRT（ディスプレイ装置）と監視盤に表示する。また，CDT より入力された情報は，一部はそのまま，一部は必要な変換を加えて他局へ転送する。

例題の業務についてまとめてみると，入力は CDT 入力，出力は CRT と監視盤（ディジタル出力）及び CDT 出力，処理はデータの変換（加工）と上限違反監視である。

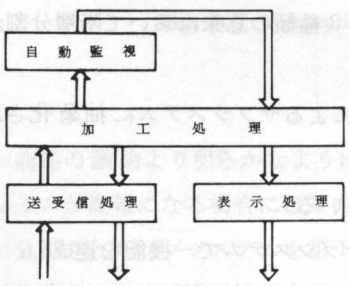
これを整理して業務フローの形に図示すると 図-9 のように描ける。

次に 図-9 を処理の複雑さによって描き直す。処理の複雑さはデータの加工度によると考えると，図-10 のようにレイヤ化フローを描ける。図-10 の個々のブロックを機能によって分



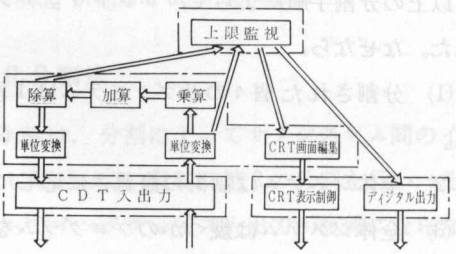
図—9 業務フロー

Fig. 9. Work flowchart for Example.



図—10 レイヤ化フロー

Fig. 10. Layered flowchart for Example.



図—11 階層化分割の例

Fig. 11. Example of hierarchical decomposition.

割し，関係線を整理すると，図-11 の分割が完成する。図-11 よりさらに細い分割を必要とする場合は再度，個々のブロックについて同様の分割を繰り返せば良いが，一般には図-11 に示した程度の分割で一つのタスクを構成する。図-11 は明らかに前節に示した意味で階層化がなされている。

8. あとがき

本稿では，電力系統制御用計算機のソフトウェアにおける階層化分割の必要性とその目的について説明し，階層の概念とプログラムモデルの定性的な検討結果から，記述，統合，保守(変更・増設)が容易で，高信頼性が実現可能な階層化分割の一手法について提案した。

提案した手法について定量的に評価する一般的な規準は今の所存在しないが，試験的に応用した結果では，プログラム設計の初心者でも経験者の従来手法による設計と同程度の日数でソフトウェア設計が可能であった。

今後，分割されるべき個々のタスクの定量的規準，階層化分割に対する定量的評価規準についての検討を進めたいと考えている。

参考文献

1) 奈良・伊藤： 昭和51年度電気関係学会関西支部連合大会，G4-26.
2) 奈良・辻他： 昭和48年度電気関係学会関西支部連合大会，G4-32.
3) Mesarovic 他： Theory of Hierarchical, Multilevel, Systems (1970), Acadimic Press.
4) 研野和人監訳： 階層システム論 (1974), 共立出版 ((3)の日本語訳).