出力指向の段階的詳細化による設計法

織 田 健† 片 山 卓 也†

ソフトウェア開発においてプログラムを関数としてとらえると，仕様記述とは入出力データ間の関係を定義するものである．また仕様作成とは要求仕様を徐々に詳細化していく過程であると考えられる．ここで抽象的概念のまま入出力データ間の依存関係等の定義を進め，より細かい記述をする段階に入ったときに初めて必要なだけの詳細化を行う手法を用いることにより，要求仕様やデータ構造の詳細を事前に過剰な詳細化を行わずに仕様作成を進めることが可能となる．さらに，解析・設計を出力指向で行うことにより，構造不一致問題の回避に中間ファイルを必要とせず，問題を局所的に解決できる．本論文では，このデータ構造の段階的詳細化による仕様作成法 SDR 法について述べる．

Design Methodology Based on Output Oriented Stepwise Refinement

TAKESHI ODA† and TAKUYA KATAYAMA†

In software development, if we regard a program as a function, then the specifications of this program can be seen as definitions of the relationships between the input and the output data of the program. If the input and output data structures of the program and the relationships between them are stepwise refined and defined, then it is possible to start software design even if the details of the requirement and of the data structures have not yet been fully determined. Furthermore, output-oriented analysis and design on stepwise refinement can reduce the need for the intermediate files that are produced during clash handling. In this paper, we will offer a specification methodology, SDR, which approach is based on the stepwise data structure refinement strategy.

1. はじめに

近年コンピュータシステムの規模の拡大によりソフトウェアの複雑さが増大している．これにともない，ソフトウェア要求の解析・明確化に非常に多くの時間が必要となっている．また仕様書と実行コードの整合性の確認も困難となっている．このため仕様作成過程の形式化と実行コードを生成可能な仕様記述言語が必要である．

前者の問題に対して．構造化設計技法[1]や JSP 法[4),5]等のソフトウェアの設計法に関する研究が盛んに行われている[8]．しかしこれらの研究は，プログラム構造の細部の設計にまで形式的に定義したものは少なく，それを定義した JSP 法等においても，目的となるソフトウェアに関する詳細な要求がすでに出来上がっていることを前提とし，その要求仕様を定められた形式に従って記述したのちに実際の解析もしくは設計が進められている．ところがソフトウェアが大規模化するに従い要求そのものが多様化・複雑化し，詳細な要求を決定すること自体が困難となってきた．

一方，プロセスやデータの解析・階層化を合理化するために，プロセス分解の過程を機械的計算により求める手法として Adler らの代数を利用したモジュール分解法[7]や荒井らによる CASDS[6] 等が提案されている．しかしこれらのプロセス分解方法は，個々の要素間の詳細な関係を示す行列を用意し，この行列から計算内容の依存関係を解析してプロセス分解を行うものであり，取り扱うデータ量が増大しデータ間の関係が複雑になると，設計者に理解しにくいプロセス分解が出力結果となる可能性が高くなる欠点があった．

本論文では，前述の問題に対処するため，段階的詳細化によるデータ構造の決定と，データ間の依存関係の定義・解析により，仕様作成を行う手法 SDR 法* を提案する．この SDR 法は，事務処理などに代表される入出力データ間の計算が静的に定義できる問題領域を対象とした純関数的設計手法で，局所性と読解性が高いプログラム設計が可能である．また出力指向で

† 東京工業大学工学部情報工学科
Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology.

* specification methodology based on Stepwise Data structure Refinement

段階的な解析方法を取ることで，要求が完全に固まっていない段階から設計を開始しても，データの抽象的概念間の関係から適切なプログラム構造が得られる．SDR 法では計算モデル HFP に基づく専用の仕様記述言語 Ares を用いるが，これによって記述された仕様からは，同様に HFP に基づいた関数型言語 AG のプログラムが得られる．

2. 基本方針

2.1 仕様のとらえ方

入出力データ間の計算が静的に定義できる問題領域では，リアリティブ・システムのような，外部からの刺激や時間による計算内容の変化は存在しない．このような静的な計算を行うプログラムに対する要求は，与えた入力から希望する出力を求めるものとなり，プログラムはその演算を行う関数としてとらえられる．

プログラムを関数とみなしたとき，設計とは要求から入出力データの構造と意味を明確にし，その間の関係を関数として定義する過程である．この関数は多入力多出力関数となる．しかし要求から導かれる入出力データ間の関係は一般に複雑であるため，この関数で定義すべき複雑な関係をいくつかの小さな関係に分解し，それらおのおのを一段階小さな要求と考え設計を続ける．この繰り返しの結果得られる関係定義はまさに仕様記述であり，前述の特徴より，その仕様記述言語のモデルには階層的関数型計算モデル HFP[10] が有効である．

2.2 プログラム構造の決定

HFP に基づく仕様記述言語を用いた設計において，プログラム構造の決定は個々のステップでモジュール分解の決定に当たる．このモジュール分解を決定する際に設計者が考慮すべき問題は入出力データ間の依存関係である．つまり個々の出力データを計算するのに必要な入力データ（このとき両者の間に依存関係があるという）を求め，共通の内容をもつもしくは関連が深い内容をもつ計算をまとめて一つのモジュールに割り振る．ここで相互に関連深い出力データ計算は，同じ入力データを必要とする傾向があるため，依存する入力データの共有度に着目したグループ分けにより，規則的にモジュール分割を得られる．設計におけるこれらのモジュール分解の役割を考えると，それは複雑な要求を徐々に小さな要求に分解し，明確化する作業であり，すなわちこれは要求解析に相当する．

この要求解析を体系的に行うため，入出力データ間の依存関係を，出力データ名とその計算に必要な入力データ名の集合と考える．そして入力データ名の共有度が高い出力データをまとめることでモジュール分解を決定する手法が考えられる．しかし一般に入出力データの構造は複雑で，そのすべての要素名を列挙すると，これらの関係は非常に巨大で繁雑なものとなってしまう．またデータの依存度の大きさを正確に定義することも困難なため，ここから機械的に得られるモジュール分解は設計者の希望からかけ離れたり，演算の意味が付けにくくなる場合が多い☆．また一度にすべてのモジュールが平面的に生成されることとなり，階層的なプログラム構造ではなくなってしまう．

この問題の原因はデータ構造中のすべての要素を一度に平面的に列挙してしまったことにあり，「データ構造は設計過程において徐々に詳細化する」ことにより解決できる．データ構造を木構造で表現すると，個々の頂点はそれをルートとする部分木中の要素の概念を代表する抽象データであると考えられる．この抽象度の高いデータ間の依存関係をもとにしたモジュール分解は，すべての要素を列挙する手法に比べて圧倒的に容易である．この分解を繰り返すことにより，階層的なモジュール分解を得ることができ，また同じ概念をもつ計算は一つのグループに分類されるため，モジュールの計算の意味がより明確なものとなる．このデータ構造の詳細化は個々のモジュールに対する要求を解析する段階において，必要に応じて行うべきである．

2.3 仕様決定と並行な設計

段階的詳細化による設計手法では，その設計過程において入出力データ構造を段階的に詳細化しながら，その間の依存関係を用いて階層的なモジュール分解を得る．またその途中段階において，データ構造に関する必要以上に詳細な情報を必要とせず，また過度な詳細化は問題を複雑にするので避けるべきである．これは，ソフトウェアに対する要求の細部が定かでない状態でも，設計を進めることが可能なことを意味する．つまり段階的な設計手法では，「仕様作成」と「解析・設計」を並行して行うことが可能であり，仕様を明確化し，記述する過程が設計過程そのものとなる．

2.4 コード生成

設計が完了した後，手作業によりコーディングを行うのであれば，人間の誤りによるエラーが含まれる可

☆ M. Adler による機械的なプロセス分解の研究[7]においてこの現象を確認することができる．

能性が高まる．しかし，形式的に設計を進めればその記述からオブジェクト・コードを生成でき，誤りの混入防止が可能であり，ソフトウェアの品質向上のためにコード生成は不可欠である．

コード生成まで考慮した段階的設計手法では，その「仕様の煮詰め」と「設計」の並行性により，仕様が細部まで決定した時点とほぼ同時にターゲット・ソフトウェアを得られることとなる．これはソフトウェア開発の合理性向上に関して，非常に有効な性質となる．

3. 仕様記述言語 Ares

データ構造の段階的詳細化に基づく仕様作成法「SDR 法」では，仕様記述言語 “Ares” を用いて仕様記述を行う．この Ares は，前章の段階的設計の基本方針に従い，属性文法に基づく関数型計算モデル HFP を基本とする言語である．

ここでは本論文で設計例として用いる「電報解析の問題」を紹介した後，Ares について述べる．

3.1 電報解析の問題

例題とする問題は「電報解析の問題」[2] といい，その概要は次のとおりである．

目的　個々の電報に関する報告書を計算する．

入力　電報のテープ
　ブロックで構成される．ブロック中の文字列は任意個の空白で区切られた単語の並び．各電報の区切りは特別な単語 “ZZZZ” で表現する．入力の終了は空電報で示す．

出力　電報に対する報告書
　各電報に対して，総単語数と長過ぎる（13文字を越える）単語数を報告．

ただし SDR 法による設計は詳細な要求が与えられていない状態から行えるため，設計初期には「電報のテープから，個々の電報に対する報告書を作成する」程度の抽象的な要求のみが得られていると仮定し，要求解析・設計が進むにつれ要求自身も詳細化されるとする．

3.2 Ares の概要

Ares は SDR 法による設計を効率よく行えるように導入された言語である．Ares は計算モデル HFP に基づく入出力間の関係定義と抽象的概念記述を中心とする言語で，多入力多出力の階層的関数型プログラムの記述能力をもち，以下の項目を目的とする．

1. データ構造の段階的詳細化が可能．
2. データ間の依存関係からモジュール分解が決定可能．
3. プログラム構造と同様の構造をもつドキュメントの記述が可能．

これらの目的のうち 1. と 2. を実現するため，入出力データ間の依存関係の有無を表す行列を用意する．これを I/O 表と呼ぶ．

また3. を実現するため，Ares の構造に Hypertext System[3] の概念を導入した．Hypertext System とは，情報を細分化してそれぞれをノードに格納し，個々のノード間にリンクを張ってその間の関係を記述することにより情報を管理する情報管理システムである．

ソフトウェアのドキュメントの論理構造は，プログラムの論理構造と非常に密接な関係にあり，ドキュメント内部にも階層構造が存在する．しかし従来のドキュメント管理においては，ドキュメントはいくつかの項目に分けられた文書の集合でしかない．Ares では記述される項目の内容に合わせた専用のノードを数種類用意する．これらをシートと呼ぶ．細分化されたドキュメントはそれぞれの項目に合ったシートに記述され，各シートはその間のリンクによりプログラム構造およびデータ構造に沿った階層構造をもつ．

以降 Ares で使用されるシートについて述べる．

3.3 I/O sheet

データ（Ares は属性文法に基づくため，以降データを属性と呼ぶ）の意味とその構造を表現するシートが I/O sheet（図1）である．

I/O sheet には，属性名とその属性に関する自然言語による概念が記述される．さらに ‘Sub attribute’ の欄には構成要素となる属性名が，‘Using module’ の欄にはこの属性を扱っているモジュール名が自動的にリストアップされる．このモジュール名と対応する Module sheet との間にはリンクが張られ，相互に参照可能となる．‘Type’ の欄は，その属性が基本データ

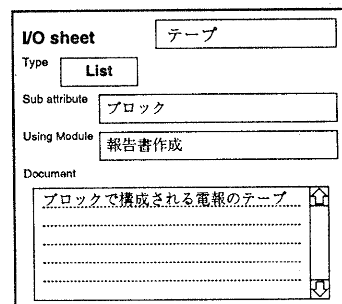


図 1 I/O sheet
Fig. 1 I/O sheet.

型（整数，実数，文字列）であるとき，その型を記述する．また特に属性の構造が列型☆であるときは，この欄に“List”を指定することで表現する．これ以外は場合は空欄となる．

シートに書かれた属性が構造をもっていると判明した場合でも，このシート内には構成要素に関する記述の追加は行わない．データ構造の詳細化により構成要素が明確になった時点で，その新たな構成要素それぞれに新しい I/O sheet が用意され，構成要素に関する概念はこの新しいシートに記述する．親となった属性を記述した I/O sheet には構成要素名がリストアップされ，親の属性のデータ構造を示す．新しい I/O sheet の生成時に，新しいシートと上記の構成要素名との間にリンクが生成される．図1は属性“テープ”に対する I/O sheet で，すでに設計が進み構成要素や構造，使用モジュールが明らかになった状態のものである．

こうした構築された I/O sheet の集合は，構成要素名と I/O sheet 間に張られたリンクにより木構造を形成する．この木構造がデータ構造そのものを表現する．

3.4 Module sheet

モジュールに関するドキュメントとその階層構造を表現するシートが Module sheet（**図2**）である．

このシートの書式は I/O sheet に類似しており，個々のモジュールに関して，“モジュール名”，“モジュールに関する自然言語による概念”を記述する．また，“このモジュールに対する入出力属性名”，“子モジュール名”がそれぞれ ‘Input’，‘Output’，‘Sub module’ の欄にリストアップされる．

Module sheet に記述する概念は，このモジュールでの計算の意味を説明するものであり，関数の概念を示している．

また属性名と子モジュール名のリストに列挙される名前にはそれぞれ対応する I/O sheet，Module sheet へのリンクが張られている．

図2は例題における最上位モジュール“報告書作成”に対する Module sheet で，すでにモジュール内の設計が終了し，子モジュール（モジュール名“報告書追加”）が得られた状態のものである．

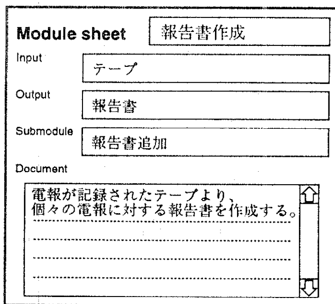


図 2 Module sheet
Fig. 2 Module sheet.

☆ リスト構造を表現する型

3.5 I/O map sheet

入出力属性間の依存関係を表す行列とそれに付随する情報を具体化した“I/O 表”を記述するためのシートが I/O map sheep（**図3**）である．

I/O map sheet は前述の Module sheet が生成されると同時に用意されるが，常に一対一対応となっているわけではない．“**case** モジュール分解”(3.5.3 項)を行う場合には，個々の条件に対して一つの I/O map sheet が用意される．

3.5.1 I/O 表

SDR 法における設計は入出力属性間の依存関係を中心に行う．このために入力属性を行，出力属性を列とする行列を用い，行列の各要素によって依存関係の有無を表現する．さらにこの行列に属性の詳細化の様子や，モジュール分解の評価対象から除外する属性の指示等を加え，設計に必要な情報の表現・操作を行えるような拡張したものが I/O 表である（図3中央）．I/O 表中の記述内容は以下のとおりである．

- 入力属性名およびその要素名

  モジュールに与えられる入力属性名が列挙される．詳細化により明確になった入力属性中の要素は，入力属性名の右側にその要素名を列挙する．図中では in. 1～in. 3 が入力属性名，e1～e3 が in. 1 の要素

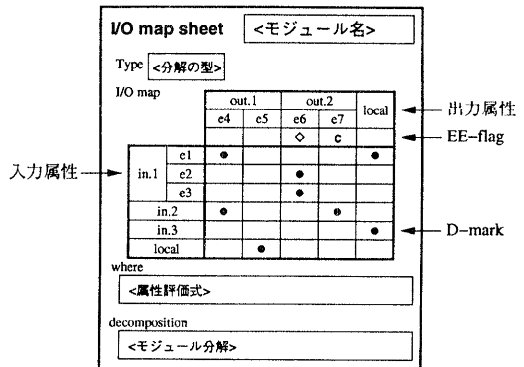


図 3 I/O map sheet の例
Fig. 3 Example of I/O map sheet.

名である.

- 出力属性名およびその要素名

モジュールで計算すべき出力属性名が列挙される. 詳細化により明確になった出力属性中の要素は, 出力属性名の下側にその要素名を列挙する. 図中では out. 1, out. 2 が出力属性名, e4, e5 および e6, e7 がそれぞれ out. 1, out. 2 の要素名である.

- 局所属性名

局所属性 (モジュール内で局所的に使用される属性) を用いる場合は, I/O 表中の入出力両方に現れる名前として表現する. 図中では local がそれに当たる.

- D-mark (dependence mark)

入出力データ間の依存関係を表すマーク. "●" で示される. I/O 表が生成された時点では, 親モジュールでモジュール分解した結果を用いてマーク付けがなされている.

- EE-flag (equation expressibility flag)

出力属性値の計算が, 入力属性 (および局所属性) と基本演算子☆から成る属性評価式☆☆によって表現可能であることを表すフラグ. "◇" もしくは "c" で示す. "◇" を使用する際には, 同時にその属性の計算方法を示す属性評価式を与える必要がある. "c" は "◇" の特殊な場合で, "◇" ＋属性評価式 "出力属性名＝入力属性名" (単純な複写) を意味する. このとき, 属性評価式は省略可能である. EE-flag が付けられた出力属性はモジュール分解の対象から外され, 属性評価式を用いてこのモジュール内部で計算される. すべての出力属性に EE-flag が付けられたモジュールは, それ以上分解されることはない.

3.5.2 その他の記述内容

I/O map sheet のその他の記述内容は次のとおりである.

- モジュール名
- モジュール分解の型

  Normal, case, for, while のいずれかを選択
- I/O 表
- モジュール分解

☆ 算術演算子 (＋, －, *, /), 論理演算子 (*and*, *or*, *xor*, *not*), 関係演算子 (==, !=, 〈, 〉, 〈=, 〉=), 列演算子 (Lisp の car, cdr, cons 等に類似) (∧, ～, ＋) 等.

☆☆ "〈出力属性名〉＝〈入力属性名と基本演算子から構成される中置形式の計算式〉" の形で記述される, 出力属性値の計算方法を表現する式.

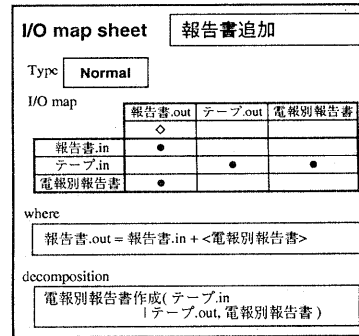


図 4 例) モジュール "報告書追加" の I/O map sheet
Fig. 4 Ex.) I/O map sheet of module "報告書追加".

I/O 表の解析結果から得られたモジュール分解の結果が, "ModuleName (InputList|OutputList)" の形式で列挙される.

- 属性評価式のリスト

I/O 表で "◇" の EE-flag を付加した出力属性に対してはそれに対する属性評価式をここに記述する.

たとえば, モジュール "報告書追加" における I/O 表は**図 4** に示すようになる (導出過程は 4.4.1 項で述べる). これは入力属性 "報告書. in", "テープ. in", 出力属性 "報告書. out", "テープ. out", および局所属性 "電報別報告書" の依存関係を表している. さらに "報告書. out" には EE-flag が付加され, その計算内容が where 以下に記述されている☆. よって子モジュール "電報別報告書作成" で計算される属性は "テープ. out", "電報別報告書" の二つとなる.

3.5.3 I/O map sheet の型

I/O map sheet には指定されるモジュール分解の型により四つのタイプがある. 図 3 で示される Normal, これに条件式の記述欄を加えた case, 同じく分解パラメータの記述欄を加えた for, 同じく分解条件の記述欄と三つの I/O 表を備える while の四つである. それぞれの型における計算の意味は次のとおりである.

Normal 分解条件や繰返し構造を含まない単純なモジュール分解.

case 条件式の評価結果によりモジュール分解を選択.

for 子モジュールへのブロードキャスト構造を構成.

while 子モジュールのパイプラインを構成.

☆ 〈…〉は長さ 1 の列を作る構成子である. この式は, 列型の入力属性 "報告書. in" に "電報別報告書" を要素として加えたものを, 出力属性 "報告書. out" の値としている.

for 型におけるブロードキャストとは，繰返し構造をもつ入力属性中のすべての要素をそれぞれ子モジュールに適用し（この分配に関する式を“分解パラメータ”と呼ぶ），その計算結果を演算子“集積子”を用いた属性評価式（“集積式”と呼ぶ）により再び一つのデータにまとめる構造である．分解パラメータは予約語“in”を用いて，“要素名 in 入力属性名”の形式で表す．集積子には，子モジュールの計算結果を要素とするような繰返しデータ構造を構成する“**sequence of**”と，計算結果が整数型もしくは実数型のときのみ有効な総和を求める“**sum of**”がある．なお **sequence of** で生成されるデータ中の要素の順序には，分解パラメータに記述された繰返し構造中の要素の順序が反映される．**図 5** の例は，列型の“list. in”と“foo”を入力属性とし，列型の“list. out”を出力属性とするモジュール“example_for”の I/O map sheet において for 型を選択した状態である．ここでは分解パラメータにより“list. in”の要素の個数だけ子モジュールが生成され，その各子モジュールに局所属性“elem”を介して名要素の値が渡される☆．子モジュールの各計算結果は局所属性“result”を介した集積式により集積され，個々の計算結果を要素とする列型データを構成し，“list. out”の値となる．このブロードキャスト構造の概念を**図 6** に示す．

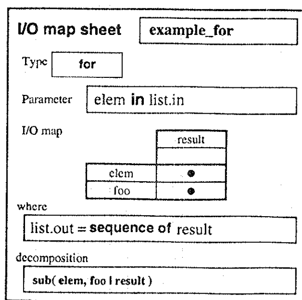


図 5 for 型の I/O map sheet の例
Fig. 5 Example of ‘for’ type I/O map sheet.

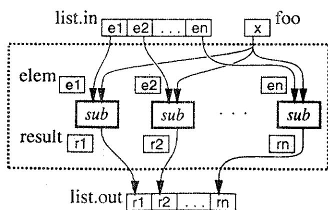


図 6 for 型の実行の概念
Fig. 6 Execution concept of ‘for’.

☆ 分解パラメータで与えられる局所属性（ここでは elem）以外の子モジュールに対する入力属性（ここでは foo）は，全子モジュールに等しい値が渡される．

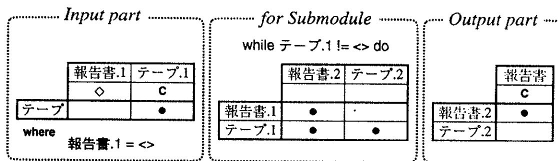


図 7 while 型の I/O 表の例
Fig. 7 Example of ‘while’ type I/O map.

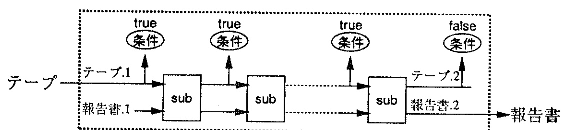


図 8 while 型の実行の概念
Fig. 8 Execution concept of ‘while’.

また while 型における三つの I/O 表はそれぞれ

- 自らの入力属性と子モジュールの初期入力属性間の依存関係
- 子モジュールの入出力間の依存関係
- 子モジュールの最終出力属性と自らの出力属性間の依存関係

を表し，**図 7** の例ではそれぞれ左から順に対応する．条件式が偽になるまで子モジュールは連結され，出力属性の値が次の子モジュールの入力属性として与えられる．したがって **while** モジュール分解における子モジュールでは，入力と出力に同じ型の属性が同じ順序で並ばなければならないという制約がある．図中では，“報告書. 1”と“報告書. 2”，“テープ. 1”と“テープ. 2”がそれぞれ結ばれ，パイプラインを形成する．このパイプライン構造の概念を**図 8** に示す．

3.6 コード生成

Ares で記述された仕様にはデータ構造，モジュール構造，属性評価式に関する十分な形式的定義が含まれており，適当なコード生成器を用意することにより，Pascal をはじめとする一般の実行可能な手続き型言語のプログラムコードを機械的に生成することができる．また Ares と同じく HFP を基本モデルとする“階層的関数型言語 AG”[9]を対象プログラミング言語とすることもでき，この場合は Ares のもつ計算機能がそのまま実装されているため，きわめて単純なコード生成器によりコード生成が可能である．

こうして得られたコードをその言語の処理系にかけ

ることで，実際に仕様を実行することができる．

4. 設計手順

本章では，SDR 法における設計手順について述べる．

4.1 要求の初期定義

設計手順の最初の段階では，出力結果として要求する出力属性とその計算に必要になる入力属性に対してそれぞれ I/O sheet を用意し，おのおののシートに属性名とその属性の“概念”を記述する．概念記述には自然言語を用い，各属性に対する説明を行う．ただし，いたずらに詳細な記述は SDR 法による設計の妨げになるため，可能な限り簡潔なものが望ましい．詳細な記述は，設計が進み具体的な構成要素等に関する計算が必要となる時点で，それらを表す sheet に記述するべきである．

例題では，入出力属性となる“テープ”と“報告書”のそれぞれについて I/O sheet を用意し，概念を記述する．

属性“テープ”「ブロックで構成される電報のテープ」(図 1. ただし型，要素名，使用モジュール名は後の設計で定まる)

属性“報告書”「電報に対する報告書」

次に，これらの属性間において行う計算内容に関する大まかな計算概念（設計初期における高抽象度の要求）を Module sheet に記述する．またこれらと同時にモジュール名を決定する．これがこれから作成するプログラムの最上位モジュールに対する Module sheet となる．

モジュール“報告書作成”「電報が記録されたテープより，個々の電報に対する報告書を作成する.」(図 2. ただし子モジュール名はモジュール分解が決定した時点でリストアップされる)．

計算概念の記述が終了すると，この最上位モジュールに対する I/O map sheet が用意される．ここで入出力属性間の依存関係を定義する（**図 9**）．

以上が設計の前段階であり，これより要求解析と設計が行われる．ここに示したとおり設計初期に与えられる要求は非常に抽象度が高いものでよく，要求自体の詳細化とデータ構造の決定が設計と同時進行する．

4.2 出力指向の解析

関数型計算モデルを用いた SDR 法における設計とは「出力値をどのように計算するか」を定めることである．したがって要求解析は常に主たる出力データに着目するところから始め，詳細化すべきデータを求める．これは常に出力データを最初に詳細化することを意味するのではなく，出力データの計算に必要なものを捜し出すことを意味する．つまり入力データは必要になって初めて詳細化される．

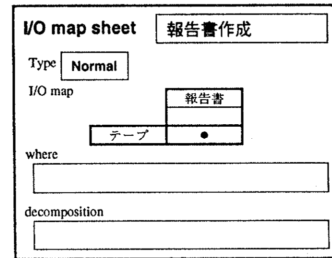


図 9 例）初期の I/O map sheet
Fig. 9 Ex.) The first I/O map sheet.

具体的には，I/O sheet や Module sheet に記述した概念記述を調べ，そこに登場する名詞はデータやその要素を表し，それらを形容する語句は繰返しの構造やデータ間の対応関係等を表現していると考えることで解析を進める．ここで着目すべき項目は

- 計算の中心となる出力属性（群）
- その出力属性およびそれが依存している入力属性の構造
- それらの要求および要素間の対応関係
- 条件により計算内容がことなる場合はその条件

等である．なお，これらの情報を効率よく抽出するため，シートに書くドキュメントはごく簡単な概念であることが望まれる．

例題では，出力属性“報告書”の概念およびモジュールの概念に着目すると，“電報”という名詞が登場し，これが計算において必要となる繰返し構造をもつデータであると考えられる．報告書はこの“電報”に対応することから，報告書も繰返し構造をもつべきであることがわかる．この報告書の要素を“電報別報告書”とする．次に，この要素の計算が依存する入力属性“テープ”に着目する．ここでも同様の解析により，“テープ”が“ブロック”の繰返し構造となっていることがわかる．こうして依存関係をもつ入出力属性がともに繰返し構造をもつことが判明する．

4.3 入出力属性の詳細化

解析の結果入出力属性の要素に関する計算が必要となった場合，詳細化によりその属性の要素を明確化する．このとき，詳細化は必要最低限のものにとどめる．無闇な詳細化は1モジュール内で処理する I/O 表を

必要以上に複雑にするためである．ただし属性評価式により直接計算できる（EE-flag を付けられる）要素についてはこの限りではない．

4.4 分解の型の選択

次に，入出力属性のデータ構造や対応関係によってモジュール分解の型を決定する．

4.4.1 繰返しモジュール分解の導入

計算の主体となる入出力データ構造の一方もしくは両方に繰返し構造が現れた場合は，**for** もしくは **while** モジュール分解の導入を検討する．

このとき入出力属性間の対応関係に着目し，双方がともに繰返し構造で，その要素が先頭から順に一対一に対応している場合のみ **for** 型を用い，その他の場合は **while** 型を用いる．

ここで **for** モジュール分解を選択した場合，入力属性をその個々の要素に落とす分解パラメータと，子モジュールでの計算結果から出力属性を構築する集積式を記述する．このとき，それぞれの要素は局所属性（4.5節）を用いて表現する．

一方，**while** モジュール分解を選択した場合は，子モジュールのパイプラインを形成し，これに必要な属性を流すことで最終的な出力属性を得る．子モジュール中で繰返し構造に関する計算を１回ずつ行う．つまりこの場合は，１段階下のモジュールと合わせて一つの繰返し構造の処理が完成するともとらえられるため，**while** 型を直接の親とするモジュールでの計算は，この時点で規定されていることになる．

例題では，依存関係をもつ入出力属性がともに繰返し構造をもつ．しかし，“電報別報告書”はその計算に必要な“電報”とは一対一に対応するが，“電報”は“ブロック”とは一対一に対応しない．したがって，このモジュールでは **while** モジュール分解を利用して“テープ”および“報告書”のストリームを作り（図7），さらに子モジュールにおいて“電報別報告書”を一つずつ計算して“報告書”を構築することになる．この子モジュールを“報告書追加”とする．

一般に **while** モジュール分解によって得られる子モジュールにおける計算は，親モジュールにおける繰返し計算に従属しており，**while** 導入時の設計の後半部分に当たり，パイプラインとして流れる属性を対象としたデータの抽出や追加を行う．したがってその解析および基本設計はすでに終了していると考えられる．つまり，例題のモジュール“報告書追加”での計算は，前述のとおり，“報告書”の要素を一つずつ求めて追加することに当たる．ここでは要素を表現する属性“電報別報告書”を導入しこれを行う（図4）．ここで属性“報告書”は詳細化され，要素“電報別報告書”に関する記述を行う．

属性“電報別報告書”「電報一つに対する報告書」

この後 D-mark の修正と，出力属性“報告書 .out”は単純な列演算により計算可能であるとの判断から，EE-flag の付加を行う（4.6節）．

次に図4の I/O 表を分割し，モジュール分解を得る（4.7節）．分割は過半数の入力属性を共有する出力属性をまとめることにより行われるが，この例では二つの出力属性“テープ .out”，“電報別報告書”はすべての入力属性を共有しているため，ただ一つの子モジュールが得られる．これを“電報別報告書作成”とする．

モジュール“電報別報告書作成”「電報一つに対する報告書を作成する．」

4.4.2 条件付きモジュール分解の導入

入出力間の対応を解析した結果条件判断が現れた場合，条件付き（**case**）モジュール分解の導入を検討する．ここで条件を入力属性による単純な式で記述できる場合は，即座に **case** モジュール分解を指定する．しかし式の記述が困難な場合は，条件判断に必要な値を取り扱う局所属性を導入し，その計算を子モジュールに任せる．

なお条件付きモジュール分解では，個々の条件に対して別々の I/O map sheet を作成・利用する．

4.4.3 単純モジュール分解

case，**while**，**for** のいずれの使用条件にも合致しない場合は，単純モジュール分解に当たり，分解型 **Normal** を選択する．

4.5 局所属性の導入

出力属性計算に関する解析により，計算に必要なデータを知ることができる．しかし，与えられた入力属性がこのデータそのものではない場合がある．このとき，必要なデータを局所属性として表現し，問題を入力属性から局所属性と，局所属性から出力属性の二つに分けることにより，計算概念から離れることなく，複雑な問題を複数のより簡単な問題に分解することができる．

ほかに以下の場合においても局所属性を導入する．

1. 条件が複雑な場合（4.4.2項）．
2. 繰返し型の要素に関する計算を行う場合（4.4.1項）．

3. **case** もしくは単純モジュール分解において，その計算に直接必要な値が入力属性と大きく異なる場合.
4. 既存モジュールの再利用のためインタフェースを合わせる場合.

なお，出力計算に必要なデータを入力属性の要素中に直接発見できなくとも，入力属性の要素のデータ構造をさらに一つ下層まで解析すると必要とするデータを発見できる場合がある（要求が境界不一致を含む場合）．この時は局所属性を導入せずに，入力属性の構造を展開する（while 型等の）モジュール分解を行う必要がある．この子モジュール以降の設計方針を定めるためのデータ構造上の1階層を越える解析を「データ構造に関する解析の先読み」という．

モジュール "電報別報告書作成" の解析・設計を例にとり，局所属性導入について説明する．ここでは計算すべき出力属性が複数存在しているため，まずモジュールの計算概念にてらし合わせ，計算の中心となる属性を探しだし，解析の焦点を絞り込む．ここでは "電報別報告書" がより重要である考え，この属性の計算方法の解析を進める．

"電報別報告書" の概念にも相変わらず "電報" が計算に必要なデータを表す名詞として登場している．ここで "電報別報告書" が依存する入力属性 "テープ.in" に着目すると，"ブロック" の繰返し構造であり，"電報" を直接の要素としてもっていないことがわかる．このような入力から "電報別報告書" を直接求めるのは困難であるため，結局計算に必要であるデータ "電報" を局所属性として導入することになる（**図 10.** ただし次節の依存関係詳細化後の状態）.

4.6 依存関係の詳細化

属性の詳細化が終了したら，これら依存関係の詳細化のために D-mark を修正する．ただし，これは詳細化前の依存関係を保存する範囲で行わなければならない．

また，その計算を子モジュールに託すまでもなく，属性評価式を用いて計算可能であると判断される出力属性については，EE-flag を立て，それが単なるコピーでなければ属性評価を記述する．

	テープ.out	電報別報告書	電報
テープ.in	●		●
電報		●	

図 10 例）モジュール "電報別報告書作成" の I/O 表
Fig. 10 Ex.) I/O map of module "電報別報告書作成".

I/O 表内のすべての出力属性に EE-flag が付いた場合 4.7 節のモジュール分解は行われない．これは，このパス* についての解析がすべて終了したこと示し，ユーザは引き続き別のパスについて設計を行うことになる．すべてのパスについての解析が終了したとき，仕様記述は完了する．

4.7 モジュール分解

依存関係の詳細化が終了したら，EE-flag のついた出力属性を除いた残りの出力属性を計算するためのモジュール分解を決定する．

モジュール分解の行うべき作業は，互いに関連の深いと考えられる出力属性がグループとなるよう出力属性を分類することである．この出力属性間の関連性を求めるために，入力属性に対する依存関係の共有の度合を判断の基準とする．われわれが利用している分解アルゴリズムは，I/O 表中の依存する入力属性を半数以上共有する出力属性を一つのグループと考え，このグループをそれぞれ一つのモジュールに割り当てるようにモジュール分解を行う．

このアルゴリズムは非常に単純であるが，SDR 法では過度の詳細化を行わずに設計を進め，依存関係が必要以上に複雑になることがないため，多くの場合，適切なモジュール分解を得ることができる．ただまれにこの基準では設計中のモジュールに与えられた全出力属性が，一つの子モジュールに割り当てる分解となる場合があるが，これは詳細化の不足を表していると判断する．

たとえば図10中の I/O 表は依存関係の共有度より，"(テープ.in|テープ.out 電報)" と "(電報|電報別報告書)" の二つに分割され，子モジュールに割り当てられる．これらをそれぞれ "電報抽出"，"報告計算" とする．

4.8 概念記述

これまでの一連の作業により一つのモジュール内における要求解析と設計，仕様記述が完了する．残る作業は詳細化により新たに生成された属性の概念記述と，モジュール分解により生成されたモジュールの概念記述である．ここでは詳細化が一段階進んでいるため，記述されるドキュメントの内容も幾分具体的にすべきである．

ここで記述した概念とモジュール分解により得られた I/O 表はそれ自体新たな要求であると考えられ，

* 最上位モジュールから現在注目しているモジュールに至るまでの経路

これらに対して4.2節以降の手順を繰り返すことができる．以上の再帰的な手続きによりSDR法における要求解析および設計過程は実現される．設計の流れの概略を**図11**に示す．

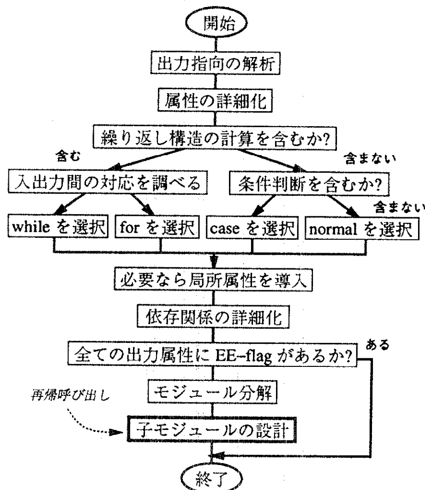


図 11 設計の流れ
Fig. 11 Outline of design flow.

```
type
  報告書 = sequence of 電報別報告書

module 報告書作成 ( テープ | 報告書 ) =
  while テープ.1 != <> do
    報告書追加 ( テープ.1, 報告書.1 | テープ.2, 報告書.2 )
  where
    テープ.1 := テープ
    報告書.1 := <>
    報告書 := 報告書.2
end module

module 報告書追加 ( テープ.in, 報告書.in | テープ.out, 報告書.out ) =
  電報別報告書作成 ( テープ.in | テープ.out, 電報別報告書 )
  where
    報告書.out := 報告書.in + <電報別報告書>
end module

module 電報別報告書作成 ( テープ.in | テープ.out, 電報別報告書 ) =
  電報抽出 ( テープ.in | テープ.out, 電報 )
  報告計算 ( 電報 | 電報別報告書 )
end module
```

図 12 例）生成されるプログラムコード
Fig. 12 Ex.) Produced program code.

4.9 例題の以降の設計

4.7節で得られた二つのモジュールについて設計を進めると次のようになる．

まず“電報抽出”では，電報の構造に注目し，それを“単語”の並びとする．一方入力属性“テープ.in”の要素は“ブロック”で，必要とする“単語”とは異なるが，さらに一階層踏み込んで解析し（解析の先読み），“ブロック”が“単語”から形成されていることを見抜く．したがって，今度は入力属性の繰返し構造を **while** モジュールの分解を用いて分解しつつ“電報”を構築する．一方，“報告計算”では，すでに一つのデータとして与えられた“電報”に関する報告内容を構築する．ここでも“電報”の構造に従い繰り返し構造が用いられる．本章で説明したモジュールに対するAGのコードを**図12**に示す．

4.10 例題のまとめ

これまで述べた例題の設計過程より，段階的詳細化により最上位モジュールに与えられた要求が徐々により簡単な要求に分解されるさまを見ることができる．

まずモジュール“報告書作成”と“報告書追加”では，**while** モジュール分解により出力属性の繰返し構造が分解されて個々の要素単位の計算に落とされる．また続く“電報別報告書作成”では局所属性“電報”が導入され，“テープ”から“電報別報告書”への計算が，“テープ”からの“電報”の抽出と，“電報”から“電報別報告書”の計算の二つに分解される．SDR法ではこのように問題の分割および簡単化により設計を進める．

5. 他の手法との比較

SDR法と同様，データ構造に着目した設計手法としてJSP法がよく知られている．また原田らにより入出力データ間の関係式の集合からモジュール分割，制御ロジックの決定等を自動化する手法[11]が提案されている．本章ではこれらの手法との比較を通してSDR法の評価を行う．

5.1 プログラム構造抽出能力

SDR法ではJSP法と同様に，データ構造と入出力間の関係からプログラム構造を抽出する．プログラム構造と選択方法に若干の差があるが，大きく異なるのは，SDR法では設計過程の前段階で全データ構造を

明確にしない点である．しかしこのような段階的設計を行っても，最終的に明らかとなるデータ構造は等しく，またデータ間の関係付けをトップダウンに行っても最終的な依存関係に差異はない．したがって入出力データ間に単純な対応関係が存在する場合，結果として等価なプログラム構造を得られる．

5.2 複数の入力に依存する出力

次に出力が複数の入力に依存している場合について考える．

このような要求に対して，JSP 法でも一応解を求められることになっているが，その解法はまったく体系的でなく，一入力一出力の場合とは比較にならないほど難解である．これは，プログラム構造を入力および出力データ構造の重ね合わせとして定義しようとしたためである．この戦略は多入力の場合はうまく機能しない．

一方 SDR 法では，依存関係のグループ分けによる問題の簡単化をプログラム構造抽出の基本としており，一入力の場合はもちろん，多入力の場合にもすべての対応付けを同様の手順で処理できる．現実の問題には複数の入力データを考慮するものも少なくなく，この特徴は大きな利点となる．

5.3 構造不一致

JSP 法では構造不一致の対応として変換器を用いた解決法を提案している．しかし変換器の体系的選択や，複数の不一致に対する対応など問題も少なくない．このような構造不一致の問題を与えられた場合の，SDR 法による設計について各不一致別に考察する．

5.3.1 順序不一致

順序不一致を SDR 法の観点から見ると，ともに出力計算に必要な要素が入力データ中にランダムに出現するととらえられる．ここでは，出力データの要素を順に計算するため，必要な入力の要素を入力データの中から逐次取り出し，必要な計算を加えるプログラム構造となる．

具体的には，入出力データをともにパイプラインとして流し☆，その子モジュールでは出力データの要素を一つ定める．この出力の要素を表す局所属性を計算するために対応する入力要素を求める必要があるが，これは順序不一致により単純な計算ではない．そこで，この必要となる入力要素を表す局所属性を導入する．こうして出来上がるプログラム構造は二つの繰返しをもち，一方で入力データから必要な要素を順に取り出しもう一方で出力データを構築する．結局単純ソートを行いながら計算を進めるものとなる．

5.3.2 入り混じり不一致

入り混じり不一致も SDR 法の観点から見れば，出力の要素を計算するにはそれに対応する入力列が必要となるが，その入力列は直接入力データ中に存在するのではなく，その要素が入力データの中に分散していることになる．これを SDR 法で設計すると，まず出力の要素を計算するために必要な入力列が入力中に存在しないので，まずこの列の構築に問題が落とされる．この入力列の構築は出力の個々の要素ごとに行われる．入力列が出来上がれば，あとは構造不一致を含まない単純な問題である．

5.3.3 境界不一致

境界不一致も SDR 法の観点から見れば入り混じり不一致とほぼ同様である．

入力列構築のために必要な要素を入力データから抽出する時点で，その要素は入力データ中の列の要素となっていることを見いだす必要がある．つまり要素の抽出のために入力データを解析するときに，別の大きな構造（この場合は列）が見えた時点であきらめるのではなく，もう1ステップ解析を進めてその要素が必要とする要素であることを突き止めなければならない．これを行わないと出力データ側の詳細化ばかりが先行して，適切なプログラム構造を得られないおそれがある．これが入り混じり不一致の場合と大きく異なる点である．

以上のことから，出力計算に必要なデータを入力データ中に直接発見できない場合は，データ構造に関する解析のある程度の先読みが必要である．

5.3.4 不一致回避の実際

前章で扱った例題は典型的な境界不一致問題であり，入力属性は2段階の，出力属性は1段階の繰返し構造をもち，その一つ目の繰返し構造間に対応関係が存在しない（**図13**）．これを出力指向で設計すると，繰返し構造間の対応が取れないことから，まず出力属性側のみの構造を分解する繰返し（SDR 法では while）を導入する．次にその要素一つの計算に必要

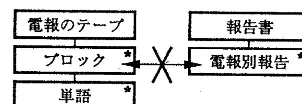


図 13 データ構造
Fig. 13 Data structure.

☆ while モジュール分解を使用．

なデータが必要となるが，入力として直接存在していないために，局所属性（ここでは“電報”）として導入される．ここでの問題は境界不一致を含んでいることから，この局所属性は入力属性の2段階目と同様の繰返し構造をもつことになる．さらに局所属性の構造を構築するために再度繰返しが導入され，当初の問題は，出力属性の要素が要求する区切りまで入力属性の2段階目にある要素を一つずつ取り出す問題に落とされる．以降は入力側を分解するような繰返しが導入される．

結局出力指向で設計を進めることにより，構造不一致を含む問題からは，出力データの構築と入力データの分解を行う繰返しを別々にもつプログラム構造が自然に得られることになる．同様の問題を JSP 法により設計した場合，そのままでは不一致により設計が行えないため，入力データ中の繰返しを解き，構造を1段階平面的な中間ファイルに変換する変換器を導入する．その後，構造不一致を起こさないこの中間ファイルを入力とする問題を解く．このとき，不一致の種類と入出力データ構造をもとに適切な構造を中間ファイルを設計する必要がある．

一方，SDR 法において二つに分けられた繰返し間の橋渡しとなる局所属性は，出力属性の要素を計算するのに必要なデータを示すものとして導入される．この段階で出力に関する繰返し構造が一つ展開されており，局所属性の計算は入力属性中からの必要な情報の抽出となる．つまり不一致の種類にかかわらず，計算に必要なデータ概念をそのまま局所属性とそのデータ構造として用いる．このとき，局所属性の構造は入力属性の構造のサブセットとなり，JSP 法における中間ファイルよりも繰返し構造一つ分簡単である．また最終的なプログラム構造は，JSP 法では全データを変換した中間ファイルを次のプログラムの入力とした形態となり，SDR 法では出力計算に必要となる要素を逐次流し込むようなストリーム形態となる（**図 14**）．ただし JSP 法によって得られたプログラムに対して，

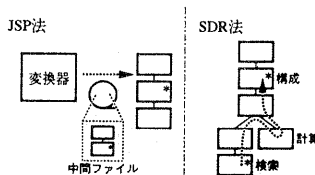


図 14 プログラム構造
Fig. 14 Program structure.

さらに変換器が主プログラムのサブルーチンとなるようなプログラム変換を行えば，結果的に中間ファイルを使用しない，SDR 法と同様のプログラム構造を得ることが可能である．しかし，その設計過程に中間ファイルの設計は依然として残ってしまう．

5.3.5 構造不一致における優位点

前節まで述べたように，SDR 法における構造不一致問題の取扱いは，出力指向の段階的設計，必要な情報を表現する局所属性の導入，入力データからの情報の逐次抽出等のアプローチにより，どの不一致に対しても同様の設計手順で対応可能である．

一方の JSP 法でも「中間ファイルの導入による問題の分割」という，ただ一つの解決方法を提示しているように見える．しかし JSP 法では，どのような構造をもつ中間ファイルを導入するか，その中間ファイルを得るための変換器はどうするか等が問題となる．また複数の不一致を含む問題では各不一致ごとに変換器が必要となり，入力データ全体を何度も変換することになるため，一つの不一致が全体に及ぼす影響が非常に大きい．

しかし SDR 法では導入する局所属性はより単純なデータであり，要求に含まれる概念を用いるため構造に関する問題もなく，また JSP 法の変換器に当たるものも単なる検索アルゴリズムにすぎない．さらに SDR 法における不一致の回避は，各不一致ごとに局所的に行われるため，複数の不一致が含まれる場合でも局所的な解決が可能である．

5.4 原田らの変換手法との比較

SDR 法同様入出力データ間の依存関係に着目し，モジュール分割，制御ロジックを決定する手法として，原田らの「非手続き的仕様から手続き的仕様への変換手法」[11]がある．

この変換手法ではまず要求仕様全体をデータ項目間の等関係式の集合として表現する．与えられた等関係式群は同じ計算範囲を持つ式をまとめてモジュール化された後，データ項目間の従属関係をもとに階層化，式の導出順序に基づく実行順序の決定，各計算式の実行条件の決定等の手順を経て手続き的プログラム☆に変換される．原田らの手法はレコードから成るファイルを処理対象としているため，データ項目間の従属関係を定義しているが，これは階層的なデータ表現と等価であり，これを用いたモジュールの階層化は SDR

☆ 実際には手続き的プログラムを生成するための SPACE 仕様

法，JSP 法と類似している．また同じ計算範囲をもつ式をまとめる手続きは，SDR 法における I/O 表の分割に当たる．

彼らの提案する等関係式は，仮想ファイルによる照合の概念が拡張された依存関係であり，JSP 法で困難な複数ファイルを用いた照合処理を含む要求を容易に扱うことができ，また SDR 法よりも簡潔に表現できる．しかし，値による条件判断後の個々の分岐における計算内容が大きく異なる場合（分岐後の依存関係に類似性が見いだせない場合）を扱えない．SDR 法では依存関係も段階的に詳細化され，このような依存関係も case 型による複数の I/O 表を用いて自然に扱うことが可能である．

また原田らの手法は，より機械的な変換手続きを与えるため，ファイルの同順条件，等関係式に関する包含条件，導出条件，階層条件，照合条件等多数の条件を満たす要求仕様のみを仮定している．しかし SDR 法では，これらの条件を満たさない構造不一致等の複雑な対応関係を含む要求をはじめ，より広い範囲の要求を扱うことが可能である．

5.5 評価

SDR 法はトップダウンで段階的な設計アプローチを取ることにより，要求の細部が明確になる以前から設計を進めることが可能であり，要求が固まると同時に設計を終えることができる．これは他の手法と比較して大きな長所である．しかも，そこから得られるプログラム構造は，単純な場合でも JSP 法のものと等価であり，構造不一致も部分木内で局所的に解決できる．よって少なくとも JSP 法と等価な能力をもち，複数の不一致を含むような大きな問題になればより優位となる．また多入力多出力を基本にしているため，現実的な問題に対してよりよい対応が可能である．

従属関係等の条件なしに階層的データ構造を扱うことができ，また入出力データ間の依存関係も段階的に詳細化するため，値に依存した複雑な計算にも対応可能である．さらに要求に対して多くの条件を仮定しないため，より広い問題領域に対して適用可能であるといえる．

6. おわりに

本論文で提案した SDR 法は段階的設計の方法論であり，これにより，詳細な要求定義・データ構造定義がないまま，設計を進めることを可能とした．さらにこの方法論がプログラム構造の設計能力において JSP 法をしのぐことを示した．また SDR 法を適応するための仕様作成言語 Ares を提案し，すべての仕様記述が終わった時点で実行コードを生成可能とした．これらにより，ソフトウェア開発過程の大幅な合理化がなされると確信する．

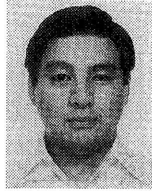
また Ares では，ドキュメントを記述したシートを Hypertext の機構により，プログラム構造にそって管理する能力ももたせ，ソフトウェアの保守性の向上が期待される．ただし，データやモジュールの構造が I/O sheet や Module sheet 上にリストアップされた構成要素名のみで表現されるため，局所的な構造把握を促し，段階的な設計を自然に行わせる効果がある反面，全体像を摑みにくくなるという欠点がある．これを補うため，視覚的に全体像を表現するブラウザなどの開発支援ツールが欠かせないものとなる．これらAres の機能および若干の支援ツールは，現在，Machintosh の SuperCard 上にプロトタイプとして実現されている．

参考文献

1) Downs, E., Clare, P. and Coe, I.: *Structured Systems Analysis and Design Method*, Prentice-Hall International (UK) Ltd. (1988).
2) Jackson, M. A.: *Principles of Program Design*, Academic Press (1975).
3) Smith, J. B. and Weiss, S. F.: Hyptertext, *Communications of the ACM*, Vol. 31, No. 7, pp. 816-851 (1988).
4) Cameron, J. R.: *JSP & JSD: The Jackson Approach to Software Development*, IEEE Computer Society Press, Silver Spring, Maryland (1983).
5) Williams, L. G.: *A Process Modeling Exercise*, Technical Report, Software Engineering Research (1988).
6) Arai, M., Matsumura, K. and Mizutani, H.: An Application of Structural Modeling to Software Requirements Analysis and Design, *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 4, pp. 461-471 (1987).
7) Adler, M.: An Algebra for Data Flow Diagram Process Decomposition, *IEEE Transactions on Software Engineering*, Vol. 14, No. 2, pp. 169-183 (1988).
8) 大林正晴，松浦佐江子，中里淳子：概念による設計法 (dmc) に基づくプログラム開発環境の実現，ソフトウェア工学，Vol. 58, No. 6, pp. 41-48 (1988).
9) 片山卓也，篠田陽一，菊池　豊，鈴木正人，今泉貴史，石原博史，高田　勝：自分自身のためのプログラム言語の作り方，*Computer Today* 別

冊，サイエンス社 (1988).
10) Katayama, T.: HFP, A Hierarchical and Functional Programming, *Proceedings of the 5th International Conference on Software Engineering*, pp. 343-353 (1981).
11) 原田 実，二方厚志：非手続き的仕様から手続き的仕様への変換—SPACE 仕様への変換手法—，電子情報通信学会論文誌，Vol. J72-D-I, No. 4, pp. 262-271 (1989).

(平成5年2月4日受付)
(平成5年9月8日採録)



織田　健（正会員）
1964年生．1988年東京工業大学工学部情報工学科卒業．1990年同大学院理工学研究科修士課程（情報工学）修了．1993年同大学院博士課程修了．工学博士．同年電気通信大学電子通信学部電子情報学科助手，現在に至る．ソフトウェア工学，設計理論，プログラミング方法論，要求解析の研究に従事．日本ソフトウェア科学会会員．

片山　卓也（正会員）
1939年生．1962年東京工業大学理工学部電気工学科卒業．1964年同大学院理工学研究科修士課程修了．工学博士．同年日本IBM（株)入社．1967年東京工業大学工学部助手，1974年助教授，1985年教授．1991年北陸先端技科学技術大学院大学情報科学研究科教授，現在に至る．ソフトウェアプロセス，ソフトウェアデータベース，ソフトウェア開発環境，ソフトウェア方法論，プログラミング言語，関数型プログラミング，属性文法などの研究を行っている．日本ソフトウェア科学会理事長．電子情報通信学会，ACM, IEEE 各会員．