Vol. 26 No. 11　　情　報　処　理　　Nov. 1985

25周年記念論文

データ抽象向けプログラム設計技法†

久　野　　靖††

データ抽象の概念はプログラムの作りやすさ，および保守しやすさを高める上で有効であるが，既存のプログラム設計技法をそのまま適用したのではその利点を十分活かすことができない．そこで構造化設計法をもとにデータ抽象の概念を取り入れたプログラム設計技法 (Extended Structured Design，以下 ESD と記す) を考案した．ESD による設計は①対象物の選択，②データ流れ図による機能記述，③モジュール構造および階層構造の決定，④対象物の実現方法の選択，⑤モジュール間の受け渡しの決定，⑥抽象データ型対象物の内部設計，の各段階を経て行われる．その特徴は①対象物を中心に設計を進め，その実体は必要に応じて抽象データ型により実現すること，②階層構造を考慮して設計を行うこと，および③設計の段階が明確に規定されていることである．本論文では ESD の具体的な設計過程について，山崎の共通例題を用いて説明する．また，他の設計技法との比較を行い，学生を対象とした試用の経験についても述べる．

1. はじめに

最近，プログラム言語にデータ抽象ないしオブジェクト指向の考えが取り入れられるようになった．従来の考え方では手続きとデータ構造を分けていたのに対して，これらの考え方ではデータとそれに対する操作を不可分のものとして扱う．プログラム言語にこれらの新しい考え方を取り入れることにより，プログラムをより局所性が高く，安全で，読みやすく保守しやすいものとすることができる．

しかるに既存のプログラム設計技法（構造化設計法など）は，そのまま適用したのでは新しい言語の利点を十分に活かすことができない．そこで，この種の言語の特徴を活かした設計が行えるような，新しいプログラム設計技法を開発することが望ましい．本文はそのような設計技法の一案を示すものである．具体的には構造化設計法[1),2)]を出発点として，これをデータ抽象の考えが活かせるように拡張する．以下この新しい設計法を仮に拡張版構造化設計法 (Extended Structured Design, ESD) と呼ぶことにする．

以下，第2章で例題を用いて ESD による設計について具体的に説明する．また第3章では，他の設計技法との比較を行い，使用経験について述べ，今後の課題について論じる．

† A Data Abstraction-Oriented Program Design Methodology by Yasushi KUNO (Department of Information Science, Tokyo Institute of Technology).

†† 東京工業大学理学部情報科学科

2. ESD による設計

2.1 設計段階と例題の説明

ESD による設計は次に示す6つの段階を経て進める（ただし，後の段階でうまく行かない所が出てきたときには，積極的に前の段階に立ち戻ってやり直す）．

① 対象物の選択
② データ流れ図による機能記述
③ モジュール構造および階層構造の決定
④ 対象物の実現方法の選択
⑤ モジュール間の受け渡しの決定
⑥ 抽象データ型対象物の内部設計

以下，1984年4月開催の設計法シンポジウムで使われた共通例題[3)]を使用し，各段階を追って設計の実施例を示す．この例題は，酒類販売会社の倉庫管理を題材としている．倉庫には複数銘柄の酒瓶を混載したコンテナが搬入され，その詳細は積荷票として受付係に渡される．一方受付係は顧客から注文を受け，どのコンテナからどれだけ搬出するかを記した出庫指示を作成する．また在庫不足の場合にはその旨記録する．課題は受付係の仕事を自動化するシステムを設計することである．

例題の問題文には明記されていない点が種々あるが，Unix オペレーティングシステム上で動かしてみることを想定して，ここでは次のように定める．

- システムの構成．受付係は手もとの端末から積荷票，出庫依頼の情報をシステムに入力する．対話は標準入力ファイル(キーボード)，および標準メッセー

ジファイル（端末画面）を用いて行い，出庫指示および在庫なし連絡の指示は標準出力ファイル（プリンタ）に打ち出す．在庫不足リストは計算機内のファイルに累積する．在庫状況は，システム起動時にファイルから読み出し，終了時にファイルに格納する．

- 入出力の形式．受付係は毎回の操作のはじめに入庫，出庫，終了処理のいずれかを指令で選択し，続いて必要な情報を入力する．出力は問題文に記されたとおり

出庫指示：注文番号，送り先名
（コンテナ番号，品名，数量，
空きコンテナ搬出マーク）…

在庫不足リスト：送り先名，品名，数量

という形式とする．「注文番号」は注文に対する一連番号で，システム内部で生成する．

- 倉庫の検索アルゴリズム．ここでは「見つかったものから出荷する」という最も単純な方法をとるが，アルゴリズムの変更は倉庫モジュールの修正だけでできるように設計する．

2.2 対象物の選択

対象物とは問題の解を得る上で必要となる「もの」をいう．対象物は能動的に動作を行う側面および受動的に操作を受ける側面を持つ．

設計の第1段階では，対象物を列挙したリストを作成する．一般にプログラムは現実世界に関する情報を処理するので，対象物は現実世界の実体に対応することが多い．したがって問題が自然言語で記述されている場合には，そこに現れる名詞が対象物の候補となる[4]．他の記述方法が採用されている場合でも，必ず「もの」に相当する項目があるはずなので，それを対象物の候補とすることができる．

次に作成したリストを検討し，問題を解く上で必要がないと判断したもの，および物理的制約などのため扱うことが困難であると判断したものを除外する．逆に，仮想的に存在するとみなすことによってシステムの働きがより自然に記述できるものがあれば，リストに追加する．最終的にリストに残った各対象物について，それが何を表すか，どんな属性・性質を持つか，どのような操作の対象になるかを記述する．

共通例題では，問題文から抜き出した名詞のリストは次のとおりである．

酒類販売会社，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，場所，コンテナ番号，搬入年月日時，内蔵品名，数量，出庫依頼(票)，電話，在庫，依頼者，コンテナ数，在庫なし連絡，在庫不足リスト，送り先名，計算機プログラム，注文番号，空コンテナ搬出マーク

これらのうち，次のものは対象物として扱う必要がないと判断されるので除外する．

- 会社，倉庫係，受付係，計算機プログラム，場所，電話．これらは外界との関係を表す言葉で，システムが自ら扱うわけではない．
- 酒．システムは個々の酒びんのレベルまで扱う必要はないと判断される．
- 内蔵品，在庫，コンテナ数．それぞれ「現在注目している銘柄の酒の集合」，「現在倉庫の中に入っているコンテナの個数」を意味するので，独立した対象物とは考えない．

入出力される対象物（下記）については仕様中に記述があるので，設計文書では説明を必要としない．

「入力」積荷票，出庫依頼
「出力」出庫指示，在庫なし連絡，在庫不足リスト，
空コンテナ搬出マーク

残りを自立した対象物として採用する．これらについて名前を整理し，必要な情報（なにを表すか，どんな属性・性質を持つか，どのような操作の対象になるか）を記述すると下記のようになる．

- 倉庫．コンテナのたまっている場所．コンテナを搬入すること，銘柄・数量を指定してコンテナの集まりを取り出すこと，中味をファイルにセーブすること，および中味をファイルからロードすることができる．
- コンテナ．酒を格納する入れ物．搬入日時，コンテナ番号を属性として持つ．銘柄を指定してその収蔵量をセットしたり読み出したりできる．また，空かどうかを調べることができる．
- 顧客（依頼者，送り先名）．取引先の名前．文字列で表せる．
- 銘柄（内蔵品名）．酒の銘柄．文字列で表せる．
- コンテナ番号．コンテナの識別番号．文字列で表せる．
- 日時（搬入年月日時）．日付と時間．文字列で表せる．
- 注文番号．注文の一連番号．整数で表せる．
- 数量．酒のビンの本数．整数で表せる．

以上の記述の中に新たに現れた名詞からも同様にして対象物を選び出す．紙面の都合上詳細な検討は省略して，結果のみを次に示す．

- 注文．〈注文番号，銘柄，数量〉の組．
- コンテナの集まり．コンテナが n 個集まったもの．コンテナを1個ずつ取り出したり追加できる．集まりが空集合かどうかを調べることができる．
- 入力ファイル．端末から文字列を入力する．
- 出力ファイル．プリンタに文字列を出力する．
- メッセージファイル．画面に文字列を出力する．
- 在庫入力ファイル．在庫状況を読み出す．
- 在庫出力ファイル．在庫状況を書き出す．
- 在庫不足ファイル．在庫不足リストを書き出す．

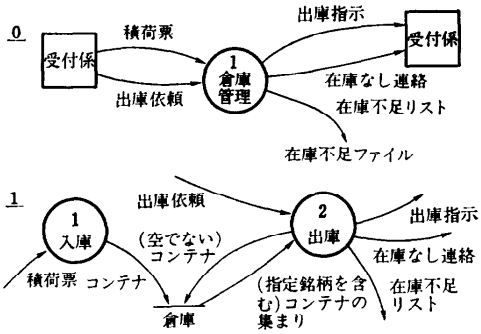
整数，文字列なども対象物と考えられるが，これらを逐一列挙することは非現実的なので，基本的データ型は対象物として記述しないという方針をとる．

2.3 データ流れ図による機能記述

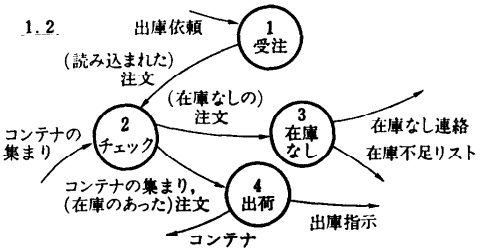
機能記述の目的は，システムが対象物の集まりをどのように扱うかを明確に定めることである．設計に用いる記法として，文献[2]の**階層化データ流れ図**を一部修正したものを採用する．その構成要素はプロセス(丸で表す)，データフロー（矢線で表す)，バッファ(水平線で表す）の3つであり，それぞれデータを処理する部分，データの流れ，データのたまっている部分を意味する．

機能記述の最初の段階は，システム全体を1つのプロセスとみなし，外界との間のデータフローを網羅した図を作成することである．これをレベル0の図と呼ぶ．続いてレベル1の図として，システムの内部をいくつかのプロセス，データフロー，バッファに分解した図を作成する．さらにレベル1の図に現れる各プロセスの内部を個別に記述する．それぞれレベル 1.1，レベル 1.2 などと名づけ，別図とする．以下同様に分割を繰り返して，各プロセスの処理が十分単純でこれ以上細かく分割する必要がない，と判断するところまで来たら処理の内容を直接記述する．その記法としては擬似コードをはじめさまざまな流儀があってよい．データ流れ図では，すべてのプロセスとバッファに対して，その役割を明らかにするような名前をつける．また，データフローに対してはそこを流れる対象物を明記する．特に，1つのプロセスで同じ対象物が2カ所以上で出入りしている場合には，それぞれの違いが明らかになるように説明を付加する*．

* 文献2)ではデータフローにも「意味のはっきりした (meaningful) 名前をつける」こととなっているが，本設計法の立場では流れるものは対象物に決まっているのでその名前を記述し，それだけではわかりにくい場合には「(どのような)何」という形で，かっこで囲んだ説明を付加することとした．



1.1
コンテナ番号，日付を入力し，空コンテナを作る．
繰り返し積荷票の終わりまで
　銘柄，数量を入力し，コンテナにセットする．
コンテナを送り出す．



1.2.1
銘柄，数量を入力し，注文を生成する．
1.2.2
もし注文に応じられる在庫があるなら
　コンテナの集まりと注文を出荷処理に渡す．
そうでなければ
　注文を在庫なし処理に渡す．
1.2.3
在庫なし連絡として，顧客名，品名，数量を出力する．
在庫不足リストとして，顧客名，品名，数量を出力する．
1.2.4
出庫指示として，
　注文番号，顧客名を出力する．
　繰り返し注文数量に達しない間，
　　コンテナの集まりからコンテナを一つ取り出す．
　　コンテナ番号，品名を出力する．
　　もしコンテナ積載数量<残り数量なら
　　　コンテナ積載数量を出力，積載数量を0にセットする．
　　そうでなければ
　　　残り数量を出力し，積載数量を残り数量だけへらす．
　　もしコンテナが空なら
　　　空コンテナ搬出マークを出力する．
　　そうでなければ
　　　コンテナを倉庫に戻す．
残ったコンテナを倉庫に戻す．

図-1 データ流れ図による機能記述
Fig. 1 Function specification by data flow diagram.

例題の対象物記述をもとに作成した機能記述を図-1に示す．レベル0の図では，システムと外界（四角い箱で示した）とのやりとりが図示されている．すなわちシステムは受付係から積荷票と出庫依頼をもらい，受付係に出庫指示と在庫なし連絡を渡す．また，在庫不足ファイルに在庫不足リストを出力する．

レベル1の図では，システムの大域的な分割が示されている．すなわちシステムは「入庫」処理と「出庫」処理に分けられ，「入庫」では倉庫にコンテナを格納し，「出庫」では倉庫から必要なコンテナをまとめて取り出し，処理した上で再び倉庫に戻すという方針をとっている．同じ対象物が行き来するため，それぞれの意味が明確にわかるように，対応するデータフローにはかっこで囲んだ説明を付している．

レベル1の記述に現れる処理のうち，「入庫」はすでにそれ自体十分単純であるので，1.1で擬似コードにより処理の内容を示してある．「出庫」についてはまだ十分単純とは言えないので，1.2以下でさらに分割している．これらを併せたものがシステムの機能記述となる．

2.4 構造図

データ流れ図がプログラムの機能を定めるのに対し，構造図はプログラムの要素間の関係を定める．記法は文献[1]を参考に拡張したもので，手続き（四角い箱で表す），対象物またはクラス（丸で表す），参照関係（矢線で表す）から成る．

ここでクラスとは「抽象データ型を定義するモジュール」を意味し，外部からのアクセスに対して保護されたデータとそれを操作する手続き（以下単に操作と呼ぶ）から成る．実際に対象物を抽象データ型により実現するかどうかは後の段階で決定するが，この段階ではすべての対象物をクラスに対応するものと考えて設計を進める．また「参照関係がある」というのは「手続き（またはクラスの操作）が他の手続き（または他のクラスの操作）を呼び出す」ことを意味する．

クラスはその性質上多くの個所から参照され，それをすべて描くことは図を不必要に複雑化させるため，参照関係の図示は手続き間に限ることにした．ただし関係を図示しない場合でも，参照する方を参照される方より上に描くという原則は守るものとする．

図-2に階層構造を記入した構造図を示す（階層構造については次節で述べる）．ここではデータ流れ図の各基本プロセス（擬似コード）をそれぞれ1つの手続きに対応させるだけで済んでいるが，一般にはこれほど簡単に行くことは限らない．データ流れ図から構造図への変換については，構造化設計法の解説書[6]に詳しいのでここでは触れないが，構造化設計法における，モジュール間の連絡（connection）を少なくし，モジュールの完結性（cohesiveness）を高める，という指針は抽象データ型の考えとよく適合する．また，データ抽象の考えを用いると，データ流れ図の規模が比較的小さくて済み，したがって構造図への変換も簡単になることが多い．

構造図中の手続きがデータ流れ図の基本プロセスに対応している場合には，手続きの処理内容はその基本プロセスの記述（擬似コード）からわかる．手続きが中間レベルのプロセスに対応している場合には，その手続きの役割はプロセス間の橋渡しを行うことである．呼び出し順序などの制御や開始・終了・例外時の処置はデータ流れ図には記述しないので，この段階で考慮する．データ流れ図からの変換が単純でない場合や制御に関して注

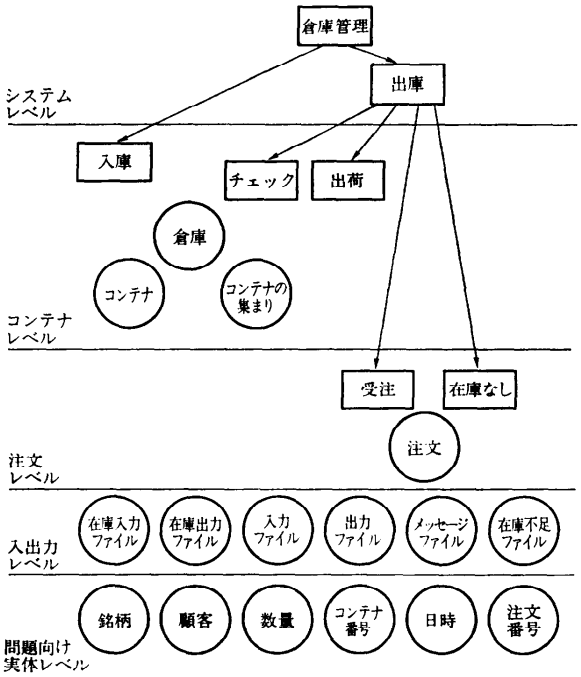


図-2 階層構造を記入した構造図
Fig. 2 Structure chart with hierarchy definition.

意すべき点がある場合には，構造図に付随する文書としてこれらを記述する．

2.5 階層構造

システムに階層構造をもたせてやると，要素間の不要な絡みがなくなり構造が理解しやすくなる．そこでESDでは，構造図の作成に引き続いて手続きと対象物の階層分けを行うことにより，設計がたしかに階層的に行われていることを確認する．具体的には，構造図の上に階層の境界線を記入し，各階層にシステム内の位置づけが明確にわかるような名前をつける．その際，類似した機能や複雑さを持つものは同じ階層に入れること，および下の階層に属するものが上の階層に属するものを参照しないことを原則とする．

階層分けを行おうとしてうまく行かない場合としては，①本来は分けることができる機能を誤ってまとめたため，どっちつかずの位置にある成分ができて境界が引けない，②ひとまとまりの機能として抽象的に考えるべきものを始めから具体的な細部まで規定してしまったため，その部分だけ不釣り合いに低レベルの成分が存在する，などが考えられる．これらはいずれも以前の設計段階における誤りであり，そのような場合には誤りのある段階までさかのぼってやり直す必要がある．

図-2に示した階層構造では，全体的制御をつかさどるもの，主として倉庫とコンテナを扱うもの，注文を扱うものをそれぞれ1つの階層とした．「チェック」，「出庫」は注文を参照するので，コンテナレベルは注文レベルより上に位置する．下位の対象物はファイル関係とそれ以外のものに分割した．これらの間には互いに参照関係がないのでどちらを上に描いてもよいが，ここでは入出力レベルを上にした．

2.6 対象物の実現

次に設計の最初の段階で選んだ対象物ごとに，実現言語の特性も考慮しながら次のうちから適切な実現方法を選択する（この段階以降ではじめて実現言語に依存した設計を行うことになる）．

① 基本型による実現．たとえば数量，個数といった対象物は標準の整数型によって実現できる．

② 複合型による実現．～の並び，～の対，などのような対象物は配列，レコードなどの定義により実現できる．

③ 仮想的実現．これは概念的には対象物を考えるが，プログラム上には対応する実体がない場合をいう．たとえば「ページ番号をつけて打つ」という仕事をする場合，概念的には「ページ」というものを考えることになるが，実現上は単に行数を数え，所定行数ごとにページ番号を打つだけで済むかも知れない．この場合，「ページ」は実体としては存在しないので，「仮想的に実現されている」という．

④ クラスによる実現．対象物を抽象データ型として定義する．この場合クラスの界面（操作名，各操作の働きと呼び出し時の情報）を規定する．呼び出し時の情報は受け渡し表（次節参照）により記述する．

この設計例では実現言語としてClu[7]を使用するが，Adaをはじめ抽象データ型をサポートする言語であれば結果はほとんど同じになる．またFortran，Pascal，PL/Iなどの言語も抽象データ機能を，言語使用上の約束を設けることによって「シミュレート」すれば使用可能である．

設計例の対象物のうち，倉庫とコンテナはその内部構造が外から見えないように抽象データ型によって実現する．そのクラス界面を**図-3**に示す（受け渡し表の読み方は次節で説明する）．コンテナの集まりの実現は，Cluでは動的な配列が使用可能なので，これを直接使用する．注文は，成分として注文番号，銘柄，数量を持つレコード型を用いる．入出力にはすべてCluの標準のファイル型であるstream型を利用し，問題向け実体はその性質から数量と注文番号は整数型，それ以外は文字列型を用いる．

2.7 受け渡しの規定

受け渡し表は手続きや操作の呼び出しの際に渡される情報を厳密に記述したもので，記法は1)を参考にしている．すなわち，呼び出し元の手続きや操作ごとに表を用意し，呼び出される手続きや操作ごとに分けて受け渡す情報を列挙する．

情報の各項目に対しては，その意味，型，入力（呼び出し元から呼び出される側に渡す）か出力（その逆）か，制御情報か否か，入力情報の場合その内容を呼び出された側で更新するか否かを明示する．設計が正しく進められていれば，機能定義，構造図，クラス界面からほぼ機械的にモジュール間の受け渡しを決定することができる．

本節の設計例におけるモジュール間の受け渡し表を**図-4**に示す．表には左から操作名，入力情報，出力情報を記してある．Cluは例外処理機構を持つので，倉庫から取り出そうとして在庫がなかったり，なんらかの原因でファイルの読み書きが失敗した場合にはシグナルによって通知する．この情報は制御情報であ

クラス stock(倉庫)

load →ファイルから倉庫の内容を回復する
save →ファイルに倉庫の状態を退避する
fetch →倉庫から指定銘柄積載のコンテナを必要なだけ搬出する
store →倉庫にコンテナを搬入する

load	倉庫ファイル名 : string	倉庫 : stock
		• not_possible : signal
save	倉庫ファイル名 : string	• not_possible : signal
	倉庫 : stock	
fetch	倉庫* : stock	並び : set
	銘柄 : string	• no_stock : signal
	数量 : int	
store	倉庫* : stock	
	コンテナ : container	

クラス container (コンテナ)

create→空のコンテナを生成する
set_quant →指定銘柄の格納数量を指定数量にセットする
get_quant→指定銘柄の格納数量を読み出す
is_empty →コンテナが空かどうかを調べる
load →ファイルに格納してあった情報からコンテナを復元する
save →コンテナの情報をファイルに格納する
get_no →コンテナ番号を読み出す
get_date→搬入日時を読み出す

create	コンテナ番号 : string	コンテナ : container
	日時 : string	
set_quant	コンテナ* : container	
	銘柄 : 銘柄	
	数量 : 整数	
get_quant	コンテナ : container	数量 : int
	銘柄 : string	
load	ファイル* : stream	コンテナ : container
		• not_possible : signal
save	ファイル* : stream	• not_possible : signal
	コンテナ : container	
get_no	コンテナ : container	コンテナ番号 : string
get_date	コンテナ : container	日付 : string

図-3 クラス界面の規定
Fig. 3 Class interface specifications.

り，●マークがつけてある．また，入力として渡されたものに呼び出された側で変更を施すものは*マークによって示した．

クラス操作を呼び出す場合の受け渡し関係は，クラス界面の記述と整合している必要がある．したがって両者には重複した情報が含まれているが，クラス界面の記述がクラス操作の呼び出し関係を一般的に定めるものであるのに対し，受け渡し表は応用システムが何のためにクラスを使うかを示している．たとえば図-3において，倉庫の load 操作が「倉庫ファイル名」一般を受け取ることを示しているが，図-4 で倉庫管理モジュールから load 操作への呼び出しでは，倉庫管理システムのセマンティクスに添って「旧倉庫ファイル名」を渡すという記述になっている．

ただし，設計者が何回も類似した記述を書いたり，手作業で受け渡し関係の一致を検査することは望ましいことではない．この点で，受け渡し表とクラス界面記述の作成・検査を支援する道具を作ることが急務であるといえる．

2.8 クラス内部の設計

ここまででシステム全体に関する記述について述べたが，クラスもそれぞれが1つの閉じた世界をなして

手続き manage_stock (倉庫管理)		
→stream		
primary_input	——	入力ファイル：stream
primary_output	——	出力ファイル：stream
open	在庫不足ファイル名：string	在庫不足ファイル：stream • not_possible：signal
getl	入力ファイル＊：stream	入力行：string • end_of_file：signal • not_possible：signal
putl	出力ファイル＊：stream メッセージ：string	• not_possible：signal
→stock		
load	旧倉庫ファイル名：string	倉庫：stock • not_possible：string
save	倉庫：stock 新倉庫ファイル名：string	• not_possible：string
→arrival	入力ファイル＊• stream メッセージファイル＊：stream 倉庫：stock	• not_possible：signal
→shipment	入力ファイル＊：stream 出力ファイル＊：stream メッセージファイル＊：stream 在庫不足ファイル＊：stream 倉庫＊：stock 注文番号：int	• not_possible：signal
手続き arrival (入庫)		
→stock		
store	倉庫＊：stock コンテナ：container	
→stream		
getl	入力ファイル＊：stream	入力行：string • end_of_file：signal • not_possible：signal
putl	出力ファイル＊：stream メッセージ：string	• end_of_file：signal • not_possible：signal
→container		
create	コンテナ番号：string 日時：string	コンテナ：container
set_quant	コンテナ＊：container 銘柄：string 数量：int	

図-4 受渡し表（一部）
Fig. 4 Interface specification tables (part).

いるので，その内部はこれまでに述べて来た一連の図などを用いて同様に記述できる．ただし，クラスは外部から呼び出される操作を複数持つので，主手続きが複数個となる．

問題の種類にもよるが，クラスの内部はシステム全体に比べればずっと単純であり，図表なども簡単なもので済むことが多い．クラス内部の設計は外部とは独立に進めることができるが，外部の他のクラスを参照することはあり得る．逆に，外部のクラスや手続きがクラスの内部事情によって影響されることは避けるべきである．

クラス内部の設計例は紙面の都合上省略するが，前

の段階で規定したクラス界面を守る限り，クラスの内部実現はこの段階で自由に決めることができることに注意すべきである．たとえば倉庫の実現は状況に応じて1次元の表，ハッシュ表，インデックス付きファイルなどのうちから適切なものを選ぶことができる．

2.9 実現段階以降

以上で①作成するモジュールのリスト，②各モジュールの外部仕様，および③各モジュールの機能記述が得られ，実現に取り掛かることができる．また，これらの設計文書は抽象度の高いものから具体的なものまで順に構造化されているので，参照や変更時の差し替えが容易である．作成されるプログラムについてもデータ抽象の考え方を活かしたものができやすく，したがって修正や拡張が行いやすい．

ここに述べた設計例をもとに実際にプログラムを作成し，Unix（バークレー版）上の Clu 処理系により実行させてみた．クラス内部の実現は最も単純なやり方として，コンテナはレコードの配列を用いた表として銘柄ごとの数量を保持し，倉庫は「コンテナの集まり」をそのまま使う，というものを採用した．プログラムのソース行数は約300行，完成した設計をもとに直接エディタでコーディングするのに約4時間を要した．虫（不良）はすべて打ち間違いなどに起因する単純なもので，修正に要した時間は30分未満であった．

3. 討　論

3.1 構造化設計法との比較

最初に記したように，ESD は構造化設計法に抽象データないしオブジェクト指向の考え方を取り入れることにより，新しい言語にも適した設計法を目指したものである．これをもとの構造化設計法と比較してみると，新しい言語に適応したこと以外にも次のような特徴がある．

- 設計手順の明確化．ESD では構造化設計法よりずっと細かく設計段階が分かれている．これにより，設計者は次に何を決めるべきか迷わずに，段階を追って設計を進めることができる．
- データ主導の設計．対象物を中心として設計を進める，という方針は設計の各段階において設計者に先へ進むための手掛かりを与える上で有効である．
- 設計の分割．構造化設計法では設計は論理的には1つの大きな構造図として表わされていたが，ESD ではクラスの内部を別に扱うことにより設計を分割し，一時に扱う問題の範囲を小さくしている．

3.2 ワーニエ法，ジャクソン法との比較

ワーニエ法[8]およびジャクソン法[9]では，最初の段階で処理されるデータの構造を下位の部分まで規定する．そのようにした場合，後から下位のデータ構造を変更することは通常困難である．一方 ESD では対象物の内部構造はできるだけ後で決める方針をとっており，特にクラスの内部構造は後で取り替えても他の部分を変更する必要がない．たとえば今回の例題で倉庫の内部に銘柄別のインデックスを別に用意して検索の効率を高めたとしても，他のモジュールはそのままでよい．

また，より新しいジャクソン開発法[10]では現実世界の実体に対応したエンティティに注目して設計を進める点が，ESD に類似している．ただし，そこでいうエンティティは後で順次実行プロセスに対応させられる手続き的なものであり，動的に生成されたり受け渡されたりするものではないので，そのデータ抽象機構としての働きは限られてれている．

3.3 試用経験

ESD の実用性を測る試みとして学生（学部3年生）を対象に，文献12)に掲載されているものと同一仕様のエディタを Clu 言語の使用を前提として設計させる課題を出し，同時に ESD について100分程度の説明を行った．その結果，37名中26名の学生が（少なくとも部分的に）ESD を使用して設計を行った．その設計文書の量はA4判レポート用紙で平均約17ページであり，うち平均約6.5ページがデータ流れ図による機能記述（クラス内部を含まない）であった．筆者の見積と照らし合わせてみると，データ流れ図の量は適切であると考えられが，記述全体の量としては40ページ程度を予想していたので，かなり少なかったことになる．これは，学生にプログラム設計の経験が乏しく，きちんと細部まで記述する習慣ができていないことによるものと思われる．設計文書の内容や品質も個人ごとに大きな差があった．

ESD を使用した学生の主な意見としては

- 手順や記述形式が明確なので設計を進める際に記法などで迷わなくて済む
- 記述が標準化されるため，他人が見ても分かりやすい設計ができる
- 抽象データ型言語についての経験が少ないにもかかわらず，その特徴を活かした設計ができた
- 階層構造は何のために決める必要があるのか分かりにくかった

などがあり，全体的には好評であった．最後の階層構造については，学生の場合プログラミング経験が少ないため「プログラムが正しく動く」という点に注意を奪われがちで，整った構造がもたらす利益についてはあまり意識していないことから，このような意見が出たものと思われる．

3.4 今後の課題

図表などをより見やすいものとすること，設計手順をさらに明確にすること，マニュアルを整備すること，計算機による支援を取り入れること，より大規模な問題に適用することで実用性に関する裏付けを行うことが今後の課題である．

謝 辞 研究の機会を与えて下さり，また貴重な助言を頂いた木村泉教授に感謝します．

参 考 文 献

1) Stevens, W.P., Meyers, G. J. and Constantine, L. L.: Structured Design, IBM Syst. J., Vol. 13, No. 2, pp. 115-139(1974).
2) DeMarco, T.: Structured Analysis and System Specification, Prentice-Hall, New Jersey, p. 352(1978).
3) 山崎：共通問題によるプログラム設計技法解説，情報処理，Vol. 25, No. 9, pp. 934 (Sep. 1984).
4) Booch, G.: Software Engineering with Ada, Benjamin Cummings, California, p. 504(1983).
5) Meyers, G. J.: Reliable Software through Composite Design, Mason/Charter, New York, p. 159(1975). 久保，国友訳：高信頼性ソフトウェア—複合設計，近代科学社，東京，p. 181 (1976).
6) Yourdon, E. and Constantine, L. L.: Structured Design, Prentice-Hall, New Jersey, p. 473 (1979).
7) Liskov, B. et al.: CLU Reference Manual, Lecture Notes in Computer Science 144, Springer-Verlag, Berlin, p. 190(1981).
8) 鈴木：構造化プログラミングワーニエ・メソド，情報処理，Vol. 25, No. 9 pp. 946-954 (Sep. 1984).
9) Jackson, M.A.: Principles of Program Design, Academic Press, London, p. 299(1975).
鳥居訳：構造的プログラム設計の原理，日本コンピュータ協会，東京，p. 318(1980).
10) Jackson, M. A.: System Development, Prentice-Hall, New Jersey, p. 418(1983).
11) 久野，遠城，木村：データ抽象化言語 CLU による中規模システム作成の経験，情報処理学会ソフトウェア工学研究会資料 22, pp. 43-48(1982).
12) カーニハン，プローガー原著，木村 泉訳：ソフトウェア作法，共立出版，東京(1981).

(昭和60年7月1日受付)