ソフトウェア工学 140－7
（２００３．３．６）

レガシープログラム変換法の実務プログラムへの適用

梶尾 義規† 魚田 勝臣‡ 永田 守男†

†慶應義塾大学大学院理工学研究科 〒223-8522 横浜市港北区日吉 3-14-1
‡専修大学大学院経営学研究科 〒214-8580 川崎市多摩区東三田 2-1-1
E-mail: †{kajio, nagata}@ae.keio.ac.jp, ‡uota@isc.senshu-u.ac.jp

本研究の目的は，企業が保持している一括処理方式の COBOL のレガシープログラムを，クライアント／サーバや Web 技術を利用した新しい形のプログラムに自動変換する手法の提案である．これまでに，一括処理を行う COBOL プログラムを対象とし，一件別処理のオブジェクト指向 COBOL プログラムに自動的に変換するアルゴリズムを提案した．これを実際に企業で使用されている COBOL プログラムに適用し，変換法がある程度有効であることを確かめた．変換法と，実験に用いた COBOL プログラムの内容，実験結果について述べる．

The evaluation of transformation method for legacy programs with application to business programs

Yoshinori KAJIO† Katsuomi UOTA‡ and Morio NAGATA†

Faculty of Science and Technology, Keio University 3-14-1 Hiyoshi, Yokohama, 223-8522 Japan
School of Business Administration, Senshu University 2-1-1 Higashimita, Kawasaki, 214-8580 Japan
E-mail: †{kajio, nagata}@ae.keio.ac.jp, ‡uota@isc.senshu-u.ac.jp

This paper proposes an automatic method transforming batch processing legacy programs into web-based, client server modeled object-oriented classes. It transforms the programs into Object-Oriented COBOL (OO-COBOL) classes, which process transactions interactively. We evaluated this method by applying it to COBOL programs currently used in business. This paper describes the method, feature of programs used for evaluation, and result of the experiment.

1. はじめに

本研究の目的は，集中型，オフライン一括処理を行う手続き型のプログラムを，オンライン一件別のオブジェクト指向プログラムに自動的に変換し，さらにクライアント/サーバ環境で動作する Web プログラムに自動的に変換する方法の確立である（図 1）．

これまでに，一括処理の COBOL プログラムを元に，クライアントとサーバに分かれ，クライアント側は MVC パターン，サーバ側は EJB のトランザクション処理モデルに対応した構造のオブジェクト指向 COBOL（OO-COBOL）プログラムを生成する方法を考案し，実装を行った[1,2]．

変換の方針，規則などを検討するにあたっては，COBOL システムの資料に載っているサ

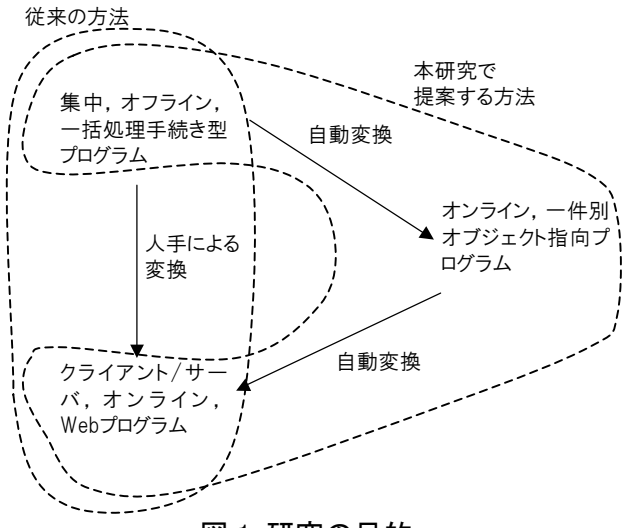

図 1 研究の目的

ンプルのプログラムを使用した[3]．このため考案した方法が企業で実際に使用されているCOBOL プログラムに対して適用できるか確かめることが課題であった．今回システムインテグレータ１社の協力によりその機会を得た．以下，変換のアルゴリズムと実務プログラムを使用した実験について述べる．

2. サンプルを元に考案したプログラム変換のアルゴリズム

プログラムの変換は，2 段階に分けて行う（図 2）．まず，意味付与などの前処理をした後，手続きの流れを一括処理から一件別処理に変えながらクライアント側とサーバ側の 2 種類のクラス･プログラムに変換する（変換 1）．次いでこの 2 つのクラス･プログラムを，クライアント側は MVC の 3 種類のクラス・プログラムに，またサーバ側は Session と Entity の 2 種類のクラス・プログラムに変換する（変換 2）．

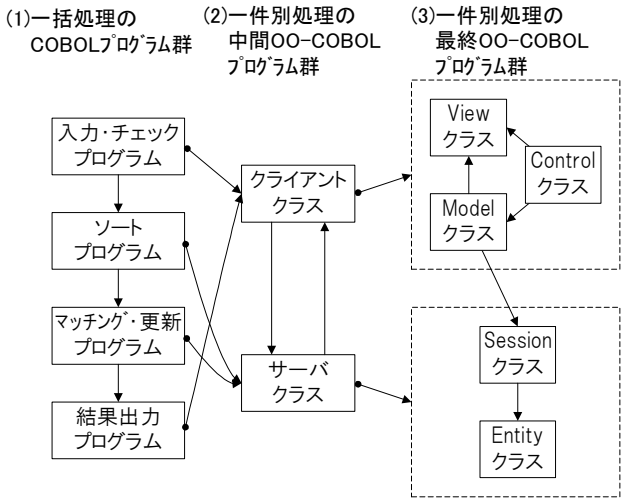

図2 サンプルを用いて検討した2段階の変換とプログラムの対応づけ

2.1 プログラムへの意味の付与

意味の付与とは，プログラムやプログラム内のデータの宣言，ひと続きの手続き等の要素が,一連の処理の中で果たす役割を検出し，その要素に対して役割を表すラベルを付与することである．

意味の付与は，次のことを行うための準備として必要である．

(1)COBOL プログラムをフレームワークに対応づけて変換するための，プログラムとそのデータや手続きの役割の判断．
(2)プログラムの手続きを一括から一件別に変えるための，繰り返しなど一括処理のロジックの検出．

予め個々のプログラムとデータや手続きに意味のラベルを付与しておき，変換を行う際に要素を抽出するための手がかりとする．

プログラムやデータの役割，特定のロジックなどの要素は，構文のパターンだけでは判別できないため，意味の付与によって補う．

2.2 変換 1

変換 1 では，図 2 に示した 4 種類の COBOL プログラムを，クライアント側とサーバ側の 2 種類の OO-COBOL クラスに変換する．クライアント側クラスは，画面の入出力，入力データのチェックなど,ユーザ･インタフェースの機能を持つ．またサーバ側クラスは，トランザクションのマッチング･更新の機能を持つ．

変換では，予め 2 つのクラスの構造と，必要なメソッドのスケルトンを持つテンプレートを用意し，そこに COBOL プログラムから情報を抽出・変換してあてはめる方式を採る．

(1) クライアント側クラスの生成

クライアント側クラスは，画面から入力データを受け付けてトランザクション･レコードに格納し，チェック処理を行い，正しいデータはチェック済トランザクションに格納してサーバ側クラスに渡す．この処理を行う属性とメソッドをクラス内に生成するために，入力･チェックおよび結果出力プログラムから要素の抽出・変換を行う．

(2) サーバ側クラスの生成

サーバ側クラスは，クライアント側クラスからトランザクション･レコードを受け取り，マスタ・レコードとのマッチングを行った結果が正しければ，追加，更新，削除などの処理を行う．ソートおよびマッチング･更新プログラムから要素の抽出・変換を行う．

2.3 変換 2

変換 2 では，変換 1 によって生成したクライアント側クラスを MVC パターンに対応付けて 3 種類のクラスに，サーバ側クラスを EJB モデルに対応付けて 2 種類のクラスに分ける．

変換１と同様に予め５つのクラスの骨格を持つテンプレートを用意しておき，そこに変換１による OO-COBOL クラスから情報を抽出・変換してあてはめる．

(1) クライアント側クラスの MVC クラスへの分解

クライアント側クラスで行っていた処理のうち，画面表示と入力の受付けを View クラスに，チェック処理を Model クラスに，これら 2 つのクラスの制御の役割を Controller クラスにふり分ける（図 3）．

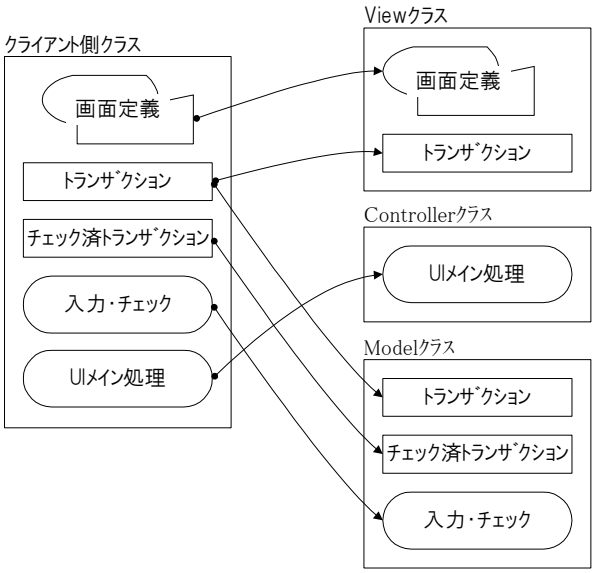

図3 View, Controller, Modelｸﾗｽへの分解

(2) サーバ側クラスの Session, Entity クラスへの分解

サーバ側クラスで行っていた処理のうち，トランザクション・レコードとマスタ・レコードとのマッチングを Session クラスに，追加，更新，削除などの更新処理を Entity クラスにふり分ける（図 4）．

この変換法について，COBOL を扱うシステムインテグレータ１社で実際に運用されているプログラムを使用して評価を行う機会を得た．実験内容について 3.で述べる．

3. 実務プログラムへの適用実験

3 つのジョブ，計 17 本のプログラムの入出力や手続きの特徴を分析し，次の 3 種類の一括処理に分けられることがわかった．

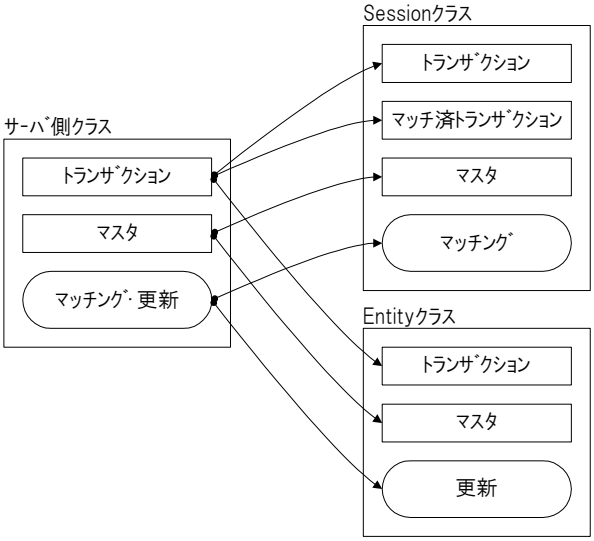

図4 Session, Entityｸﾗｽへの分解

(1) トランザクションを入力し，キーの変わり目を判定して集計を出力する処理．
(2) トランザクションを入力し，新しい項目を追加するなどレコードの形式を変えて出力する処理．
(3) うえの 2 種類にあてはまらない処理．

それぞれに該当する典型的なプログラム 1 本を抽出した．以下に実験の方針，個々のプログラムの変換の実際について述べる．

3.1 実験の方針

実験の目的と手順を次のように設定して行った．

(1) 実験の目的

サンプルのプログラムを用いて考案した変換アルゴリズムが，実務のプログラムにもあてはまるか検証する．すなわち，

・命名規則からデータの役割が判断できる．
・手続きの節，段落の呼出し関係から，個々の役割を判断できる．
・データの定義を移動する，繰返しのロジックを除去する，など要素に対する操作を行える．

主にこれらの前提があてはまるか，変換を試行して確かめる．

(2) 実験の手順

a. 変換の方針の決定

最終的にどのような OO-COBOL プログラムに変換するか検討する．①データの入出力の

変更，②手続きの一括処理から一件別処理への変更，③必要に応じて，元のプログラムのデータを組み合わせた新しいファイルの導入，などを決める．

b. 変換後のプログラムの構築

決定した方針に従って，変換して出来上がった OO-COBOL プログラムを人手で組む．

c. 変換規則の検討

元のプログラムと，b.で構築した変換後のプログラムを比較し，プログラム内の要素に対する変換の規則を検討する．

変換のプロセス中で最も複雑な操作を行っているのは，変換１で元のプログラムの入出力を変更し手続きを一括処理から一件別処理に変更する段階である．そこでこの段階について特に詳細に述べる．

3.2 一件別処理への変更

データと手続きの特徴に基づき抽出した 3 つの一括処理プログラムを，次のように一件別処理に変更し，一つのメソッドとして利用する．各プログラムについて，現行と変換後の処理内容，手続きの変換，それらに基づく変換規則を述べる．

(1) 集計，出力プログラム

a. 現行の処理内容

トランザクションをファイルから順次読み込み，小計合計を計算・出力する一括処理を行う（図 5）．このときマスタ DB から文字情報を取得して印字する．

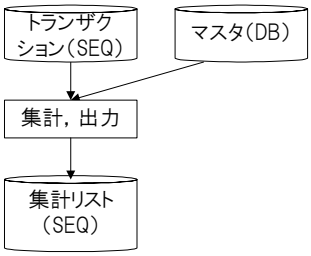

図5 現行の集計処理

b. 変更後の処理内容

トランザクションをレコードとして一件別に受け取り，随時に小計合計を計算する．計算結果は新しく作る集計ファイル（索引編成）に累積していく．照会するときは，この集計ファイルからデータを出力する（図 6）．

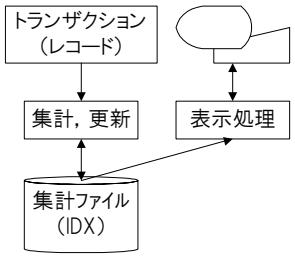

図6 変換後の集計処理

c. 手続きの変換

現行の一括処理の手続きの流れは，次のとおりである．

①トランザクションファイルから１レコード読む．
②明細の数値を各小計に加算する．
③小ブレイクならば，小計の合計を計算し，各小計と共に出力する．
④大ブレイクならば，全合計を計算し，各合計と共に出力する．
⑤トランザクションファイルが終わるまで繰り返す．

この手続きを，b.の方針に従って次の一件別の流れに変換する（図 7）．

①1 レコードが引数として入力される．
②明細の数値を各小計に加算する．
③小計の合計を計算し，集計ファイルの該当する小計レコードを更新する．
④全合計を計算し，集計ファイルの該当する合計レコードを更新する．

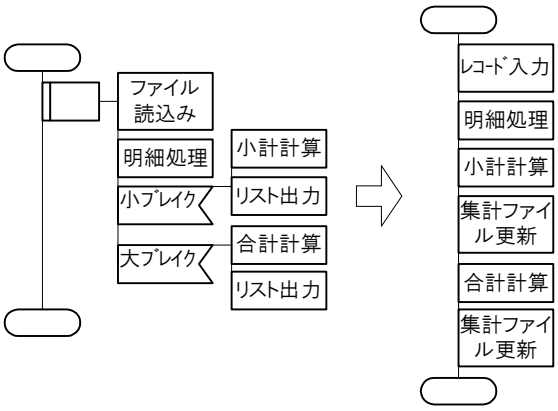

図7 集計処理の手続きの変換

d. 変換規則の検討

プログラムに以下のような変換を施す.

①一連の手続きをループする繰返しの指定を除去する.
②トランザクション・レコードの定義を，引数にするためにデータ部連絡節に移動する.
③ブレイクの判定を除去し，毎回ブレイク処理を行うようにする（図 8）.
④リスト出力の部分を，集計ファイルの該当レコードの読込み，集計項目の加算，ファイルの更新，に変える.

```
IF  NEW-BUMON  NOT =  OLD-BUMON  OR  EOF-FLG   =  'ｵﾜﾘ'
    PERFORM  BREAK-BUMON-RTN
END-IF.

IF  NEW-TENCD  NOT =  OLD-TENCD  OR  EOF-FLG   =  'ｵﾜﾘ'
    IF  MEI-FLG  =  '1'
        PERFORM  BREAK-BUMON-RTN
    END-IF
    PERFORM  BREAK-TENCD-RTN
END-IF.
```

⇩

```
PERFORM  BREAK-BUMON-RTN
PERFORM  BREAK-TENCD-RTN
```

図8 ブレイク判定の除去

(2) レコード形式変換プログラム

a. 現行の処理内容

トランザクションをファイルから順次読み込み，他のレコード形式に変換してファイルに書き出す一括処理を行う

b. 変更後の処理内容

トランザクションを一件別にレコードの形式で受け取り，他の形式のレコードに変換し，次の処理への引数として送り出す（図 9）.

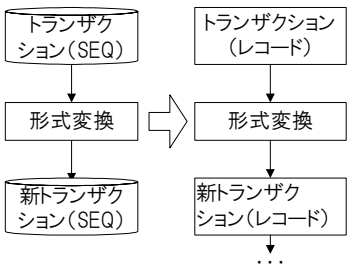

図9 レコード形式変換処理の変換の方針

c. 手続きの変換

現行の一括処理の手続きの流れは，次のとおりである.

①トランザクションファイルから 1 レコード読む.
②レコードの各項目の値を，新しいレコードに転記する.
③新しいレコードを新ファイルに書き出す.
④トランザクションファイルが終わるまで繰り返す.

この手続きを，b.の方針に従って次の一件別の流れに変換する（図 10）.

①1 レコードが引数として入力される.
②レコードの各項目の値を，新しいレコードに転記する.
③新しいレコードを次の処理を行うメソッドの引数として送り出す.

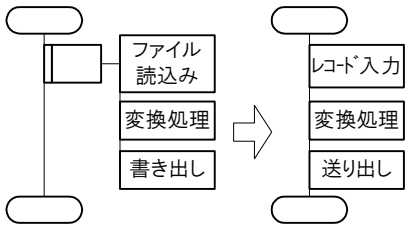

図10 レコード形式変換処理の手続きの変換

d. 変換規則の検討

プログラムに以下のような変換を施す.

①一連の手続きをループする繰返しの指定を除去する（図 11）.
②トランザクション・レコードの定義を，引数にするためにデータ部連絡節に移動する.
③新しいレコードを書き出す部分を，次の処理を行うメソッドの呼出しに変える.

```
PERFORM INIT-RTN.
PERFORM F1-READ-RTN.
PERFORM MAIN-RTN    UNTIL   EOF-FLG   =   'ｵﾜﾘ'.
PERFORM TERM-RTN.
```

⇩

```
PERFORM INIT-RTN.
PERFORM MAIN-RTN
PERFORM TERM-RTN.
```

図11 繰返し指定の除去

(3) 出力指示データ作成プログラム

a. 現行の処理内容

トランザクションをファイルから順次読み込み，同じ ID に属する下位の ID を並べて出力指示データとして書き出す一括処理を行う（図 12）.

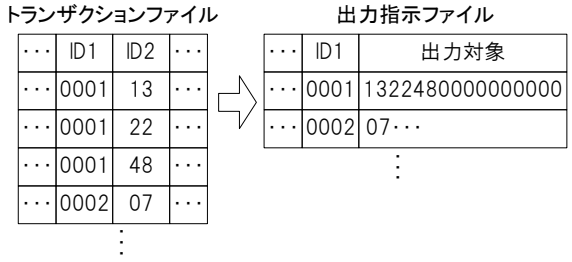

図12 現行の出力指示データ作成処理

b. 変更後の処理内容

トランザクションを一件別にレコードの形式で受け取る．出力指示データは索引ファイルに格納し，随時更新を行う（図 13）．

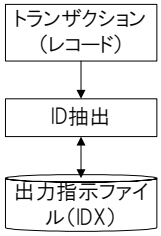

図13 変換後の出力指示データ作成処理

c. 手続きの変換

現行の一括処理の手続きの流れは，次のとおりである．

①トランザクションファイルから 1 レコード読む．
②ソートファイルに書き出す．
③全て書き出した後，ソートを行う．
④ソートファイルから 1 レコード読む．
⑤出力レコード内の配列構造に指示データを転記する．
⑥出力レコードを書き出す．
⑦ファイルが終わるまで繰り返す．

この手続きを，b.の方針に従って次の一件別の流れに変換する（図 14）．

①1 レコードが引数として入力される．
②出力レコード内の配列構造に指示データを転記する．
③出力指示データのファイルから，該当するレコードを更新する．

以上のように，考案した変換法を適用した変換の方針の決定，手続きの変換，変換規則の決定を行うことができた．

3.3 クライアント/サーバのクラスへの分解

一括処理のプログラムから一件別処理の

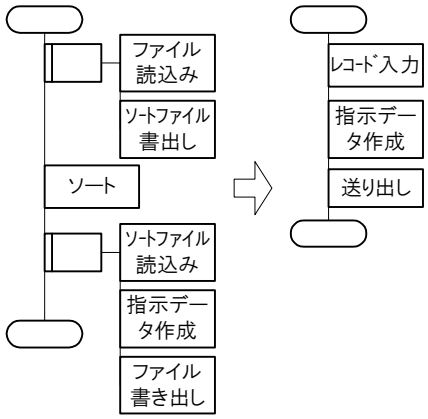

図14 指示データ作成手続きの変換

メソッドの形に変換したプロダクトのうち，計算，転記を行ってデータの内容を変更する部分をクライアント側に，最終的な結果をファイルに読み書きする部分をサーバ側に分ける（図 15）．一件別のメソッドへの変換結果を元にして，分解を自動的に行う．サーバ側の読み書きのロジックは新規のものなので，テンプレートにデータ名等をあてはめて生成する．

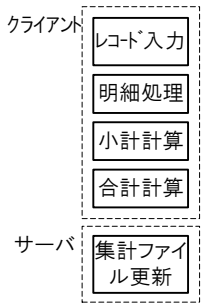

図15 集計処理のC/Sへの分解

3.4 C/S から 5 つのクラスへの分解

クライアント側にあったロジックのほとんどは Model クラスに，サーバ側にあったロジックは Entity クラスに入る．他のクラスは，データをそのまま受け渡ししたり，メソッドの呼び出し制御を行う（図 16）．C/S への分解と同様に，自動的に 5 つのクラスを生成する．またテンプレートを用いる点も同様である．

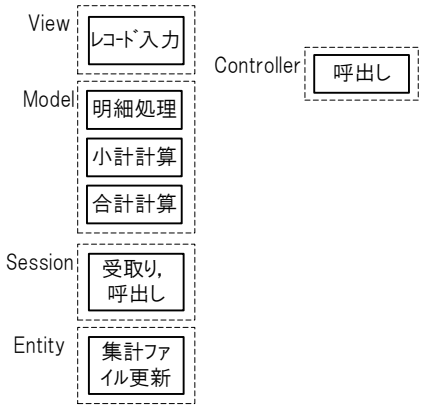

図16 集計処理の5つのクラスへの分解

4. 考察

(1) アルゴリズムの実用性

実験の目的にあげた，(1)プログラム中のデータの役割の判断，(2)手続きの呼出し関係からの役割の導出，(3)プログラム中の要素に対する操作，などが行え，考案した変換法が実務のプログラムにも適用できることを確かめた．企業ではコーディング規約を定めてプログラミングを行っていることが多く，同種の複数のプログラムに対しこれらのアルゴリズムを適用できる可能性はある．

一方，変換を行う前にまず人手で変換の方針を検討しなければならず，その手間がかかる点が本変換法の問題点である．

(2) 変換による量的な変化

変換のプロセスで，元のプログラムの内容が最も変化するのが，一括処理のプログラムから一件別のメソッドの形式に変える段階である．図 17 に，この段階で変化する行数がプログラムに占める割合を示す．形式変換プログラムの変化の割合が大きいのは，処理が単純で行数が少ないのに対し，項目数の多いレコードを移動していることが原因である．

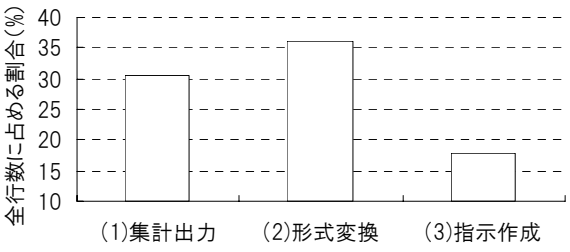

図17 一括から一件別への変更で変化する行数の割合

また，元のプログラムから C/S に変換すると行数は約 1.4 倍になり，最終的な 5 つのクラスの行数は元の 1.6 倍になる．

5. おわりに

レガシープログラムの再利用を目的とした変換法について，実務で適用できる可能性があることを示した．本方法による変換結果を利用して，さらに EJB などの枠組みに対応させて変換する方法を提案すること，またオブジェクト指向の観点で変換後のプロダクトの構造を洗練することが今後の課題である．

謝辞

本研究は，ソフトウェア工学研究財団ならびに株式会社アイネスのご協力により実施することができた．関係者の方々にお礼申し上げる．

参考文献

[1] 梶尾，魚田，永田:一括処理のレガシープログラムからオブジェクト指向プログラムへの変換法，信学技報，KBSE2002-15，pp. 13-18(2002)．

[2] 梶尾，魚田，永田:プログラム資産活用のための再構造化，FIT(情報科学技術フォーラム)2002，pp. 4-251-252(2002)．

[3] 日立製作所:COBOL85 プログラミング，6190-3-724(1995)．