Vol.2012-SE-175 No.12
2012/3/15

変更要件に関係するプログラム特定のための処理名抽出

岡田譲二[†]　　坂田祐司[†]

有識者や既存の設計書がない状況で，システムの保守開発を行うためには，業務用語とシステムの用語の対応関係を把握することが重要である．本稿では，ソースコードを処理機能という粒度で分割し，入出力されるファイル名と処理機能から機能名を自動的に付与する手法を提案するとともに，この手法を実現するツールを試作した．また，COBOL 言語で記述されたプログラムに適用し，定性的な評価を行ったところ，本手法によって業務用語とシステムの用語の対応関係が把握しやすくなったことを示した．

Processing Name Extraction for Understanding the Correspondence Relation

Joji OKADA[†]　and Yuji SAKATA[†]

Understanding the correspondence relation between the business-term and the system-term is important for maintenance of systems without experts or the existing design specifications. This paper proposes an automated method for generation of function name. The function names generated by this method are made of input/output file name and processing name extracted from source code. We have developed a function name generator. Moreover, we have applied this generator to the program described in COBOL language, and qualitative evaluated. The result shows this method is effective for understanding the correspondence relation between the business-term and the system-term.

1. はじめに

昨今の厳しい経済状況を受けて企業のシステムへの投資額は年々減少しており，システムの開発費も削減が強く求められている．特に近年では，企業の IT 投資額のうち 6 割を占める既存システムの保守開発費の削減が求められている [1].

この保守開発の作業のうち 4 割から 6 割がソフトウェアの理解のために費やされていると言われる [2]．これは，多くのシステム開発の現場では，保守開発担当者が既存システムの全てを把握しているわけではないためである. この原因としては例えば，他社が開発したシステムの保守開発を受注した，システムが巨大化・複雑化しすぎて保守開発担当者も把握できなくなった, 有識者が後進を育てる前に退職してしまった，などの理由が考えられる．また，長年保守開発されているシステムにおいては，設計書は存在しなかったり, 陳腐化していたりするため信用できない状態である事が多い.

このような状況での保守開発においては， SWEBOK [3]が挙げている保守開発に特有なアクティビティのうち,「変更依頼が分析され，システムの用語に翻訳される」に時間がかかる.

このアクティビティは，処理方式によって難易度が変わる. 企業の業務システムは，その処理方式から対話処理とバッチ処理に大別され，バッチ処理のほうが「変更依頼が分析され，システムの用語に翻訳される」は難しい.

対話処理とは，画面を介してユーザとシステムがやり取りする処理方式である．対話処理では，変更依頼に記載されている業務用語（申込書チェック，引当処理など）からソースコードに記載されているシステム用語（ファイル名, 機能名, 項目名など）の翻訳は比較的容易である．その理由としては，対話処理では業務用語を画面内の項目名などに使っている場合が多く, 業務用語と画面の対応関係をとるのは容易であり，画面から利用されているソースコード数やファイル数は少ないためである.

一方で, バッチ処理とは, 一定量のデータなどを一括して処理する方式 [4]であり，COBOL 言語などで実装されたジョブと呼ばれる処理を，JCL などの言語で順次実行する処理方式のことである．バッチ処理は，あらかじめ定められた複数の処理が一度に行われるため，調査すべきソースコード数やファイル数は多く，対応関係の調査には非常に時間がかかる．特にソースコードはファイルよりも数が多く，調査に手間がかかる.

このため，バッチ処理について業務用語と機能名の対応関係を理解する作業を支援する技術が必要である.

そこで本稿では，バッチ処理を対象として，ソースコードを処理機能という粒度に分割し，入出力されるファイル名と処理機能から機能名を自動的に付与する手法を提

[†] 株式会社 NTT データ
NTT DATA CORPORATION

ⓒ 2012 Information Processing Society of Japan

案する．

本稿の構成は以下のとおりである．まず，2 章では業務用語とシステム用語の対応関係の理解が必要な状況について例を交えながら詳細を述べる．3 章では，本稿で提案する処理機能について述べ， 4 章でソースコードから処理機能を抽出し，機能名を自動的に付与する手法について述べる．5 章で本手法を実装したツールにおける適用実験の結果を示す．6 章で関連研究を述べ，最後に 7 章でまとめと今後の課題について述べる．

2. 業務用語とシステム用語の対応関係

本章では，簡単な想定在庫数計算機能を考え，これを具体例として業務用語とシステム用語の対応関係を分析する既存の手法と，既存手法での問題点を述べる．また，既存手法での問題点を解決する方向性についても述べる．

(1) 想定する保守開発担当者の知識レベル

本稿で想定している状況は，保守開発担当者が既存システムについての十分な知識を持っていない状況である．具体的には，業務要件に記載されている業務用語とシステム用語の対応関係についての知識は持っておらず，各ソースコードがどのような内容か把握していない知識レベルであることを想定している．他社が開発したシステムの保守開発を受注した場合などは，このような知識レベルの場合が多い．

(2) 既存システムの想定在庫計算機能

本章で考える想定在庫数計算機能は，1) 生産計画ファイルに記載された「本日生産されて入荷できる予定の在庫数」と，2) 想定在庫ファイルに記載された「倉庫に昨日まであった在庫数」の 2 つの合計を本日の想定在庫数とし，想定在庫ファイルとして出力する機能である．

この機能の詳細は，以下の通りである．

- 生産計画ファイルは，「商品コード」，「生産予定日」，「生産予定数」，「生産工場」という 4 つの項目を持つ．
- 想定在庫ファイルは，「商品コード」，「想定在庫数」という 2 つの項目を持つ．
- 生産工場には， α 工場と β 工場がある． α 工場は倉庫に近いが， β 工場は倉庫からやや離れた場所に位置する．
- 現行機能では，β 工場で生産される商品は本日中に倉庫に入荷されないため，α 工場で本日生産される予定の商品だけを，想定在庫数としている．

この想定在庫計算機能のデータフロー図を**図 1** に示す．この図は，四角のオブジェクトがプログラムを，丸みを帯びたオブジェクトがファイルを，その間を繋ぐ矢印がプログラムとファイルがどのような関係にあるかをそれぞれ表している．例えば，左上の FILE-A，PGM-1，FILE-B は，

- プログラム「PGM-1 生産計画ファイル抽出」はファイル「FILE-A 生産計画ファイル」を入力とし，ファイル「FILE-B 生産計画ファイル」を出力する

ことを意味している．

また，同じ名前が付いているファイル名同士は，同じレコードレイアウトのファイルであることを意味する．さらに，各プログラム名・ファイル名は，既存の設計書の情報を元に作られているとする．

このようなデータフロー図は，ソースコードと機能一覧などの設計書があれば既存のツールで作成することができるが，既存のツールで作成されたデータフロー図は，プログラム数が数百，数千個になることも珍しくない．本稿ではスペースの都合上，関連部分のみを抜粋した．

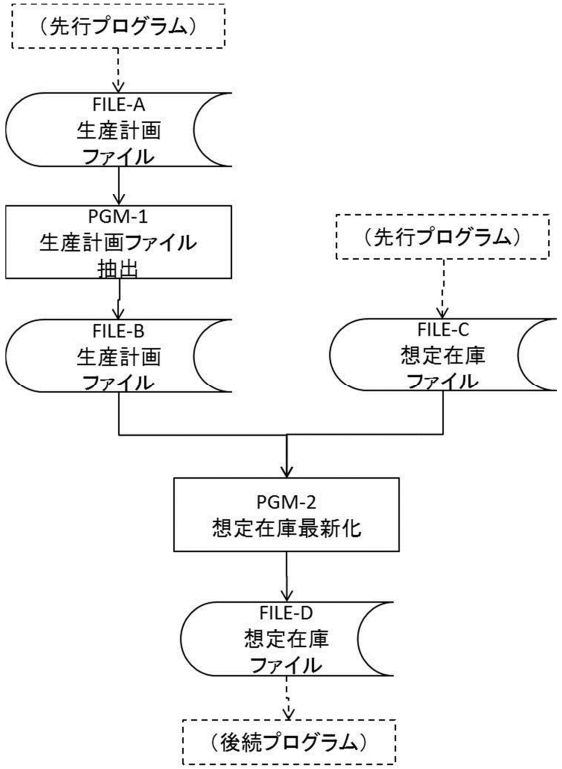

図 1 既存手法によるデータフロー図の例

ⓒ 2012 Information Processing Society of Japan

(3) **変更依頼の内容**

このような既存システムに対して，以下のような変更依頼に基づいて保守開発を行うことを考える．

- β工場から倉庫までの配送が改善され，β工場で生産される商品も当日中に倉庫に入荷されるようになった．このため，想定在庫数もβ工場の当日の生産数を考慮して算出するように変更する．

(4) **既存手法による業務用語とシステム用語の対応関係の付け方**

業務用語とシステム用語を対応付ける方法としては，変更依頼からキーワードを抽出し,それに該当するファイル名や機能名をデータフロー図から検索する方法がある．

この方法では最初に，変更依頼からキーワードを抽出する．今回の例では，キーワードとして「β工場」,「想定在庫数」,「生産数」といったファイル名に関連するキーワードや,「想定在庫数算出」といった機能名に関連するキーワードを抽出することが考えられる．

次に，これらのキーワードをもとに，データフロー図を検索し，変更依頼に関係しそうなファイル名や機能名を抽出する．具体例では．ファイル名に関連しそうなキーワードとして「β工場」,「想定在庫数」,「生産数」が抽出できる．また，これらのキーワードを項目として持つファイルを検索し,「生産計画ファイル」と「想定在庫ファイル」が抽出できる．一方で，機能名に関連するキーワードとして「想定在庫数算出」を挙げたが，これらに合致する機能名がデータフロー図内には存在せず，変更依頼から関連する機能名を把握することができない．

このため，機能名では絞り込みが出来ず,「生産計画ファイル」と「想定在庫ファイル」が関連するプログラムを全て抽出し,それらのソースコードの中身を全て読み,今回の変更依頼と関係があるかどうかを調査しなければいけない．

(5) **既存手法の問題点**

前節で述べたような手法には，既存システムについての知識がない保守開発担当者は変更依頼から既存システムの機能名に合致するキーワードを選定することができないという問題がある．

既存の機能名に合致するキーワードを選定できない原因は，データフロー図に記載されている機能名として，過去のシステム開発担当者や保守開発担当者によって，まちまちな粒度で，まちまちな名前を付けられた機能名を利用しているためである．

(6) **問題点解決の方向性**

前述の問題は，以下の 2 つの条件が満たされていれば解決すると考えられる．

- 決まった粒度，決まった命名規約で機能名が付与されたデータフロー図がある
- その命名規約を保守開発担当者が把握している

以上の 2 つの条件があれば,既存システムの知識がない保守開発担当者であっても,命名規約を参考にして，変更依頼からデータフロー図の機能名に合致するキーワードを作成することができるようになる．適切なキーワードを作成することが出来れば,該当する機能やプログラムを検索することもでき，結果として調査しなければならないソースコードを減らすことができると考えられる．

以降の章では，統一された粒度と名前を持つ機能として処理機能と呼ばれる単位を提案する．また，この処理機能を用いてデータフロー図を作成する手法についても述べる．

3. 処理機能とは

本稿では，ソースコードから処理機能を抽出し，機能名を自動付与する手法について述べるが，本章ではまず，本手法で抽出される処理機能について定義する．

本手法で提案する処理機能とは，複数のレコードからなるファイルを一括処理する決まった粒度の処理のことであり，**表 1** で示した 7 種類を定義した．

表 1 処理機能の定義

処理機能	定義
抽出	入力ファイルの一部のレコードだけを出力ファイルとして出力する
振分	入力ファイルのレコードを条件に従って，複数の出力ファイルに振り分けて出力する
マッチング	2 つの入力ファイルを特定の項目同士で比較し，(1)項目の値が等しいレコードが両方の入力ファイルに存在する，(2)項目の値が等しいレコードが片方にしか存在しない，の 2 種類に振り分けて出力する
和	複数の同じファイルフォーマットの入力ファイルのレコードを，順に出力ファイルとして出力する
作成	1 個以上の入力ファイルの値を利用して、入力とは異なるファイルフォーマットの出力ファイルを出力する
編集	入力ファイルの一部のレコードの値を変更して，出力ファイルとして出力する
集計	入力ファイルを集計し，集計キーと集計値（合計値、平均値、最大値、最小値）からなる出力ファイルを出力する

この処理機能は，データベースなどで用いられる集合を操作する関係代数の演算子を参考に作成・定義した．関係代数の演算子としては,「和」,「差」,「交わり」,「直積」,「制限」,「射影」,「結合」,「商」,「属性名変更」,「拡張」,「要約」などが知られている [5].

関係代数と処理機能の対応関係を**表 2** に示す．処理機能は，関係代数の名前を保守

ⓒ 2012 Information Processing Society of Japan

開発担当者にも馴染み深い名前に変更したり，いくつかの関係代数の演算子をまとめて一つの処理機能としたり，逆に一つの関係代数の演算子を 2 つの処理機能として分割するなどして，定義した．

表 2 関係代数と処理機能の対応

関係代数	処理機能
和	和
差	マッチング
交わり	
商	（「マッチング」と「作成」の組み合わせで表現）
制限	抽出/振分
射影	作成/編集
直積	
結合	
属性名変更	
拡張	
要約	集計

関係代数を参考にした理由は，本手法で対象としているバッチ処理が複数のレコードをまとめて処理する方式であり [4]，この複数のレコードをひとまとまりの集合として考えた場合，バッチ処理は関係代数の演算子に相当すると考えられるためである．

一方，関係代数の演算子その物を利用しなかった理由は，関係代数の演算子名と，一般的な業務システムで用いられる機能名が大きく乖離しており，一般的な保守開発担当者がキーワードとする機能名に相応しく無いと考えたためである．例えば，「ある指定した条件に合致するレコードのみを出力する」という操作は，関係代数の演算子では「制限」であるが，これを処理機能では「抽出」と表現している．

処理機能を用いたデータフロー図の具体例として，2 章で説明した在庫集計機能に対して処理機能を抽出したデータフロー図を**図 2** に示す．

処理機能を抽出した結果，既存手法では「想定在庫最新化」だけだった PGM2 が，「生産計画ファイル抽出」，「生産計画ファイル・想定在庫ファイルマッチング」，「想定在庫ファイル編集」，「想定在庫ファイル和」の 4 つとして表現されている．

このため 4 章の例では，まずキーワード選定の際に，変更依頼に記載された「想定在庫数も…考慮して算出」から，処理機能を意識して，「算出」などではなく，「編集」という動詞を考える．また，ファイル名は既存手法でも分かっていた通り，「想定在庫ファイル」を考え，先ほどの処理機能「編集」と組み合わせて，機能名「想定在庫ファイル編集」というキーワードを選定する．このようにして作成した「想定在庫ファイル編集」という機能名であれば，提案手法で作成するデータフロー図から検索することができ，今回の変更依頼に対応するプログラムは「PGM-2」であることが分かる．

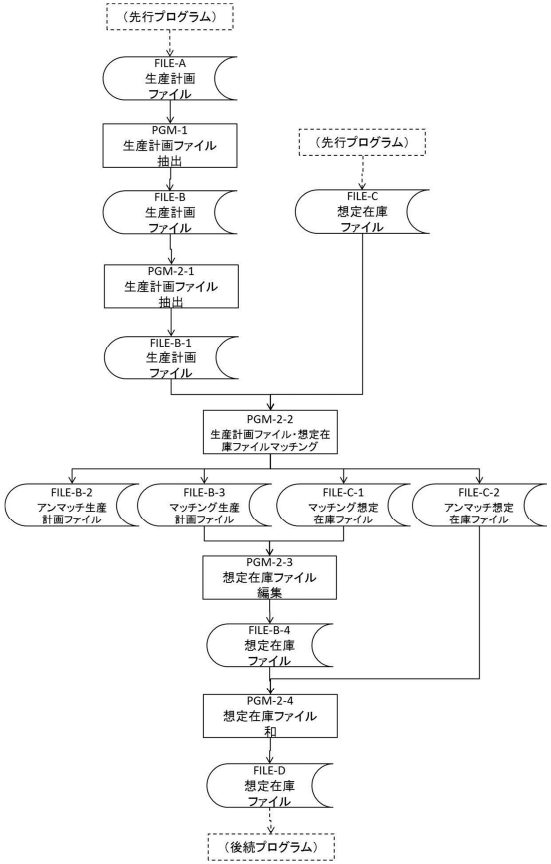

図 2 処理機能を抽出したデータフロー図の例

4. 処理機能の抽出手法

本章では，前章で述べた処理機能をソースコードから抽出し，抽出された処理機能を用いたデータフロー図を作成する手法について述べる．

ⓒ 2012 Information Processing Society of Japan

4.1 処理機能の抽出手法の概略

本手法の概略は，**図 3** に示した通りであり，大きく分けて 4 つの手順からなる．まず，ソースコードを構文解析し，抽象構文木を得る．次に，この抽象構文木を単純な要素だけからなる抽象構文木に変換する．続いて，この変換された抽象構文木に対して，予め定義しておいたマッチングルールを適用し，処理機能を抽出する．最後に，抽出された処理機能名と，処理機能の入出力ファイル名から機能名を作成する．以降ではこの 4 つの手順の詳細を述べる．

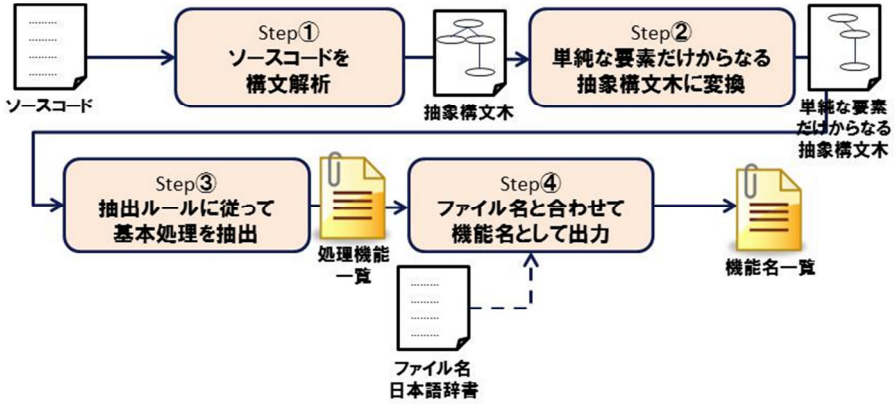

図 3 提案手法の概略

4.2 Step1: ソースコードの構文解析

Step1 では，ソースコードを構文解析し，抽象構文木を作成する．Step1 で作成される抽象構文木は，ソースコードが記述されているプログラミング言語独自の要素からなる．

例えば，COBOL 言語の「MOVE 1 TO VAR」という文は，(MOVE, VAR, 1)といった要素で抽象構文木を作成されるし，C 言語の「var = 1」という文は(=, var, 1)といった要素で抽象構文木を作成される．

4.3 Step2: 単純な要素だけからなる抽象構文木に変換

Step2 では，抽象構文木を単純な要素だけからなる抽象構文木に変換する．単純な要素としては，「順構造」，「条件分岐」，「反復」，「外部入力」，「外部出力」，「代入」の 6 種類を考える．この変換のために，Step1 で作成したプログラミング言語独自の要素が，**表 3** に記載された単純な要素のどれに該当するかというマッピングルールを作成しておき，変換の際には，このマッピングルールを用いて，単純な要素だけからなる抽象構文木に変換する．

例えば，COBOL 言語の抽象構文木(MOVE, 1, VAR)は，(=, VAR, 1)となる．

表 3 Step2 の抽象構文木に用いられる要素

単純な要素	抽象構文木での表現
順構造	Block *他の要素*
条件分岐	If *条件式 真文となる Block 偽文となる Block*
反復	Loop *反復条件式　反復される Block*
外部入力	Input *外部入力が格納される変数 外部入力元*
外部出力	Output *外部出力する変数 外部出力先*
代入	= *代入先の変数　代入元の式*

4.4 Step3: 抽象構文木に対するマッチング

Step3 では，変換された抽象構文木に対して，予め定義しておいたマッチングルールを適用し，処理機能を抽出する．

マッチングルールの一例を**表 4** に示す．

表 4 マッチングルールの一例

抽出する処理機能	マッチングルール
抽出	「変数と定数を比較する条件式を持つ条件分岐の外側の順構造に，外部入力が存在する」…① かつ 「①の条件分岐の真文 Block もしくは偽文 Block に，外部出力が存在する」…②

4.5 Step4: 機能名の作成

Step4 では，抽出された処理機能名と，処理機能の入出力ファイル名から機能名を作成する．「作成」以外の 6 つの処理機能（「抽出」，「振分」，「マッチング」，「和」，「編集」，「集計」）は，「*入力ファイル名　処理機能名*」という命名規約で命名する．また，「作成」は「*出力ファイル名　処理機能名*」という命名規約で命名する．

例えば，処理機能が「抽出」で，入力ファイル名が「想定在庫ファイル」の場合，機能名として「想定在庫ファイル抽出」と命名する．また，処理機能が「作成」で，出力ファイル名が「想定在庫ファイル」の場合，　機能名として「想定在庫ファイル作成」と命名する．

ⓒ 2012 Information Processing Society of Japan

5. 適用実験

本章では,4 章で述べた処理機能の抽出手法を実装したツールについて述べ,COBOL 言語のソースコードから，業務用とシステム用語の対応関係を理解するために有用な処理機能を抽出することができることを示す.

5.1 処理機能抽出ツール

試作した処理機能抽出ツールの概要を**図 4** に示す.

Step1 を実現するために Strafunski [6]を利用した. Strafunski は Haskell で実装された言語処理モジュールであり，SDF [7]で記述した言語仕様を利用して，対象言語のソースコードを抽象構文木（実態は Haskell のツリー構造）に変換する. Strafunski を利用した理由としては，将来的に解析対象の言語を COBOL 言語だけではなく，C 言語や Java などの多言語に対応することを考慮した際に，自ら改変可能な言語パーサを利用することが望ましいと考えたためである.

また，Step2 で利用する抽象構文木の走査・変換モジュールとしては，StrategyLib [8]と呼ばれる Haskell のツリー構造を操作することができるライブラリを利用した. このライブラリを利用すると，ツリー構造の走査・変換の条件を簡単に記述することができる.

Step3 で利用する抽象構文木のマッチングルールには，前述の StrategyLib を利用して処理機能毎に Haskell でマッチングルールを作成した.

Step4 として，マッチングにより抽出された処理機能と入出力ファイルの関係を表示するために，Graphviz [9]を利用した.

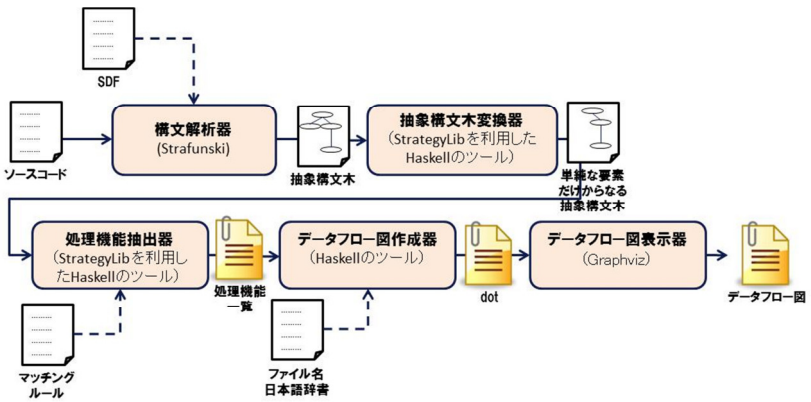

図 4 試作した処理機能抽出ツールの概要

5.2 抽出対象のプログラム

適用実験において，処理機能を抽出する対象プログラムとして，JCL と COBOL 言語で書かれた申込書チェック機能を用いた. この機能は，申込書ファイルや，それに付随する補助情報ファイル（漢字氏名ファイルなど）を入力として，申込書の各項目間の必須チェックや関連チェック，補助情報との関連チェックなどを行い，なんらかのチェックが通らずエラーとなる申込書レコードを，エラー理由と共にエラーリストとして出力する機能である.

図 5 にこの機能について既存手法で作成したデータフロー図を示す.

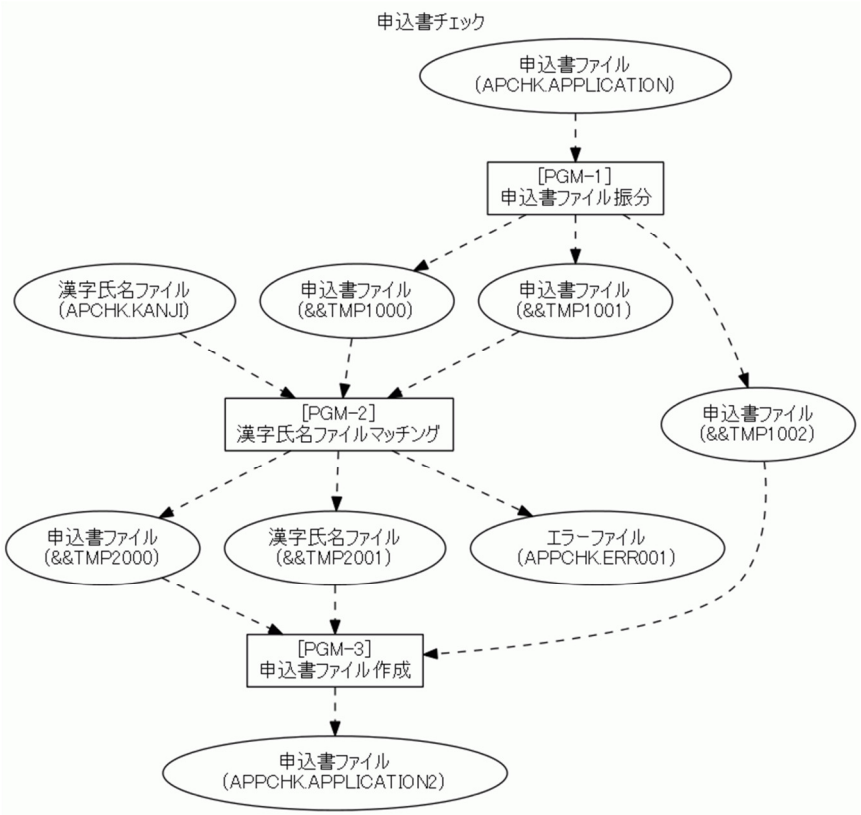

図 5 既存手法で作成した申込書チェック機能のデータフロー図

ⓒ 2012 Information Processing Society of Japan

5.3 処理機能の抽出結果

試作したツールによって作成されたデータフロー図を**図 6** に示す.

Graphviz で出力しやすいように，4 章でのサンプルと比べてノードの形を変更している．楕円形のノードがファイルを表しており，長方形のノードがプログラムを表している．また，ファイルのノードの 1 行目はファイルの日本語名を，2 行目はファイルの物理名を表している.逆にプログラムのノードの 1 行目はプログラムの物理名を，2 行目はプログラムの日本語名を表している.

処理機能を抽出した結果，**表 5** に示すように既存手法で作成したデータフロー図に比べて機能が細分化され，機能名が変化した.

表 5 処理機能の抽出による機能名の変化

ID	既存手法での機能名	提案手法での機能名
PGM-1	申込書ファイル振分	申込書ファイル振分
PGM-2	漢字氏名ファイルマッチング	申込書ファイル抽出
		申込書ファイル和
		申込書ファイル・漢字氏名ファイルマッチング
		エラーファイル作成
PGM-3	申込書ファイル作成	申込書ファイル編集
		申込書ファイル和

5.4 評価

図 6 をシステムの保守開発担当者 3 人に見せ，業務用語とシステム用語の対応関係の理解に役立つかという観点でコメントを求めた.

この結果，対応関係の理解に役立つという以下のようなコメントを得た.

- 漢字氏名ファイルの内容を申込書ファイルに転記している箇所を調べたい場合，既存手法では，漢字氏名ファイルと申込書ファイルを入力にしている PGM-2 と PGM-3 両方を調査しなければいけなかった．本手法では,「申込書ファイル編集」が PGM-3 だけであるため，PGM-2 は調査しなくてもよいことが分かる.

また一方で，影響調査の範囲も小さくできるという以下のようなコメントも得た.

- 例えば，漢字氏名ファイルの一部の項目を変更する変更依頼があった場合，既存手法では，PGM-3 において&&TMP2001 の漢字氏名ファイルと，&&TMP2000 と&&TMP1002 の申込書ファイルの 3 つが入力となって，一つの申込書ファイルを出力しているため，&&TMP2000 と&&TMP1002 の申込書ファイル両方に影響がないかを確認しなければいけなかった．一方で本手法では，&&TMP1002 の申込書ファイルは漢字氏名ファイルとは無関係なことが分かる.

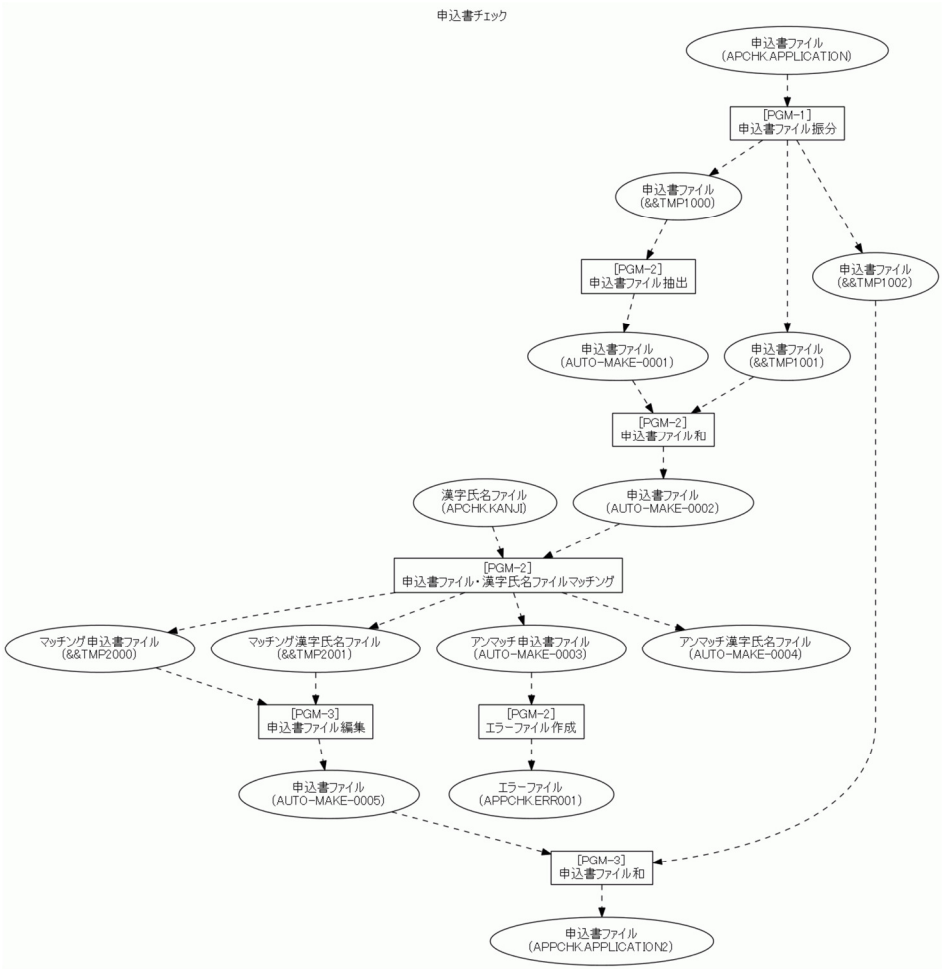

図 6 本手法で作成した申込書チェック機能のデータフロー図

ⓒ 2012 Information Processing Society of Japan

6. 関連研究

業務用語からその実装箇所を特定する研究は、実装箇所特定（Feature Location）と呼ばれ，これまで多くの研究がされてきた．Bogdan ら [10]は，実装箇所特定の研究・技術を特定技術ごとに，動的解析，静的解析，テキスト解析，それらの組み合わせに大別した．本稿の提案手法は，静的解析の一つであると考えられる．

実装箇所特定の静的解析手法として，Matin ら [11]は，オブジェクト指向言語を対象に，特徴とソースコードの対応関係を取る手法として，関心事グラフ（Concern Graph）を利用する手法を提案している．この手法では，まずプログラムを解析し，プログラムモデルと呼ばれるグラフを作成すする．続いて，プログラムモデルから、利用者が必要ないと考えるノードやエッジを取り除いて関心事グラフを作成する．本稿の提案手法とは，適用対象とグラフの粒度の 2 点で異なる．Martin らの手法はオブジェクト指向言語のプログラム一般が対象であるが，我々の手法は手続き型言語一般のバッチ処理が対象である．また，Martin らの提案するプログラムモデル及び関心事グラフのノードは，クラス，フィールド，メソッドのいずれかで，エッジは，呼び出し，読み込み，書き込み，チェック，作成，宣言，スーパークラスの 7 種類の関係のいずれかである．一方で，本稿の提案手法のデータフロー図のノードは，ファイル，処理機能のどちらかであり，エッジは，入力，出力のいずれかである．また Martin らの手法では，データと機能の関係を読み込みエッジ，書き込みエッジのどちらかで表現するが，本稿の手法では，この関係を処理機能 7 種類として表現している．

静的解析手法の別のアプローチとして，Zhao ら [12]が，ソースコードの分岐を保存したコールグラフと機能名との対応関係を利用する SNIAFL というアプローチを提案している．この手法の前提には，過去の設計者やプログラマが機能名などの識別子に意味のある名前をつけていることが必要となる．本稿の手法は，識別子の名前に意味が無い場合，あるいはプログラムの内容を表していない場合でも適用できることが異なっている．

7. おわりに

本稿では，ソースコードを入力として業務用語とシステムの用語の対応関係の理解を支援するために，関係代数を応用した処理機能を提案し，ソースコードから処理機能を抽出する手法について提案した．提案手法では，ソースコードを構文解析して得られた抽象構文木に対して，予め定義しておいた処理機能との対応関係を表すルールによって処理機能を抽出し，抽出された処理機能を元に作成した機能名を利用したデータフロー図を作成する．また，処理機能をソースコードから抽出し，自動付与された機能名によるデータフロー図を作成するツールを試作した．このツールは，COBOL 言語のソースコードから処理機能を抽出し，自動付与された機能名によるデータフロー図を自動生成することができた．さらに本稿では，試作したツールを用いて評価を行い，COBOL 言語のソースコードから業務用語とシステム用語の対応関係を理解するために有用なデータフロー図を得ることができることを示した．

今後の課題としては，本手法の有効性の検証があげられる．具体的には，被験者や対象機能を増やした本手法の定性的評価の実施と，定量的な評価の実施である．今回行った定性的な評価では，処理機能とファイル名から作成した機能名はシステムの理解に役立つという結果であったが，評価者や評価の対象機能が少ない．また実際にどの程度保守における理解工数削減に役だっているかという定量的な評価は行えていない．今後は，被験者や対象機能を増やした定性的評価を実施すると共に，今回提案した機能名一覧を利用した場合と利用しなかった場合で，どの程度の工数削減が見込めるのかを計測し，定量的な評価を行う．

参考文献

1. 社団法人日本情報システム・ユーザー協会：企業 IT 動向調査 2011（2011）.
2. Hamlen, RK Fjeldstad and W. T.:Application Program Maintenance Study: report to our respondents,Proceedings GUIDE 48(1983).
3. 松本吉弘：ソフトウェアエンジニアリング基礎知識体系―SWEBOK,オーム社(2003).
4.情報処理推進機構ソフトウェア・エンジニアリング・センター：機能要件の合意形成ガイド,http://sec.ipa.go.jp/reports/20100331/cbgfr_20100331v1.00_batch_all.pdf.
5. C.J.Date:データベース実践講義―エンジニアのためのリレーショナル理論,オライリージャパン(2006).
6. Strafunski,http://www.haskell.org/haskellwiki/Applications_and_libraries/Generic_programming/Strafunski.
7. SDF,http://www.syntax-definition.org/.
8. StrategyLib,http://hackage.haskell.org/package/StrategyLib.
9. Graphviz,http://graphviz.org/.
10. Dit, B., Revelle, M., Gethers, M. and Poshyvanyk: Feature Location in Source Code: A Taxonomy and Survey,Journal of Software Maintenance and Evolution(2011).
11. M. P. Robillard and G. C. Murphy: Concern graphs: finding and describing concerns using structural program dependencies, Proceedings ICSE '02(2002).
12. W. Zhao, L. Zhang, Y. Liu, J. Sun, F. Yang,: SNIAFL: Towards a Static Noninteractive Approach to Feature Location, ACM Transactions on Software Engineering and Methodology, Vol. 15, No. 2(2006).

ⓒ 2012 Information Processing Society of Japan