RO-002

業務プロセス自動可視化のための実践的アプローチ
A Practical Approach to Automated Business Process Discovery

矢野 啓介† 野村 佳秀† 金井 剛†
Keisuke Yano Yoshihide Nomura Tsuyoshi Kanai

1. はじめに

業務プロセス管理 (Business Process Management, BPM) 技術が大きな注目を集めている。BPM は業務プロセスの可視化、実行、監視、評価といった改善サイクルを実施するものである。したがって BPM を開始するには業務プロセスの可視化を必要とする。

しかし、現場で実施されている業務プロセスの実態を把握するには、人手によるインタビューなど、多大な手間と時間を必要とする。このことが BPM 技法適用のさまたげとなっている。この手間を削減するために、業務システムが出力するログを元に業務プロセスモデルを推定する技術も存在するが、広く活用されるには至っていない。

本論文では、ログではなく、業務システムが蓄積した業務データを活用して業務プロセスの実態を自動的に可視化する実践的な手法を提案する。本手法においては抽象化したモデルを求めるのではなく現実に実行された業務の流れを分析することに主眼を置く。本手法は日本・北米・欧州の様々な業種の業務システムへの適用を行い、成功を収めている。

本論文は以下の構成をとる。2 節では関連する先行研究を概観し、それらの特徴を述べる。3 節から 5 節では提案手法を説明する。提案手法は大きく分けて、入力データから業務フローインスタンスを得る部分 (3 節) と、得られたフローインスタンス群をフロー図として提示する部分 (4 節)、さらに例外フローの分析 (5 節) とからなる。ついで、6 節では提案手法を実際の業務システムに適用した事例を紹介する。最後の 7 節はまとめである。

2. 関連研究

業務システムが残したログを入力として元の業務プロセスを推定する技術は以前より研究されている。

初期の研究としてはAgrawalら[4] やCookら[5] のものがある。前者はノイズに対する対策がある。後者はログからのモデルの推定を有限状態機械に関連付けて論じている。Herbst [8] は機械学習の手法による並列を含むワークフローモデルの推定を論じている。Hwang ら[9] は、アクティビティインスタンスが開始日時と終了日時の両方を持つものという仮定を置くことで、アクティビティのオーバーラップを手がかりとして前後関係を推定している。

また、Aalstら[2] の提案した手法はαアルゴリズムという名称で知られている。こうした技術はワークフローマイニングあるいはプロセスマイニングと呼ばれることがある。Aalstら[1] はワークフローマイニングの研究のサーベイをまとめている。

†富士通研究所 Fujitsu Laboratories

Mansら[10] はプロセスマイニングを医療分野に応用した事例を紹介している。Aalstら[3] はオランダの公的機関の業務に適用し、プロセスモデルだけでなく、組織の分析も試みている。これらにはProMというプロセスマイニング用のツール[6] が使われている。

Ogasawaraら[12] は、ログを入力として、並行分岐と条件分岐とが組み合わされたモデルを推定する技術を開発している。

これらの研究はいずれも、ログを入力としてプロセスモデルを推定することに主眼が置かれている。

3. ログでなく業務データからの業務プロセス抽出

本節から 5 節までに、本論文の提案手法を述べる。著者らは、業務システムに蓄積された業務データを入力することによって、そのシステム上で実行された業務プロセスの実態を自動的に可視化する技術を開発した。この手法およびそれを実装したツールを BPM-E (Business Process Management by Evidence) と呼ぶ。

従来研究の多くは入力としてログを使用していた。典型的にはワークフローエンジンのログが対象となる。ワークフローエンジンが出力するログには通常、フローインスタンスのIDと、実行したタスクの種別と、そのタスクを実行した日時とが記録されている。この 3 つの情報を用いると可視化は容易である。フローインスタンスIDによって案件ごとのフローインスタンスを取り出すことができるので、各案件のタスクの流れを重ね合わせれば良い。ワークフローログからフローを抽出する例を図 1に示す。

日時	インスタンスID	タスク
2011/2/1 9:00	A0001	Task A
2011/2/1 10:02	A0001	Task B
2011/2/1 10:05	A0002	Task A
2011/2/1 11:21	A0001	Task C
2011/2/1 11:25	A0003	Task A
2011/2/1 11:30	A0002	Task B
2011/2/1 11:40	A0003	Task C
2011/2/1 12:20	A0002	Task C

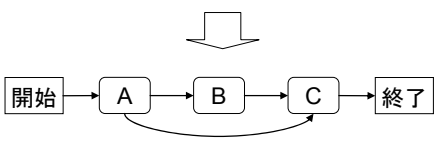

図 1 ワークフローログからのフローの抽出

しかし、現実には、業務システムがワークフローエンジンの上に構築されていることは一般的に期待できない。ワークフローエンジンのログを前提とする限り、可視化技術の適用範囲が大きく狭められてしまうことになる。

また、ワークフローのログでなくシステム的なログを用いることも考えられる。しかしこの方法では、システムの動作ログから業務上のどのようなタスクに対応するかを推定しなければならなくなる。また、複数システムをまたがっての可視化が難しくなる。

そこで本手法では、ログよりもむしろ、業務システムのデータベースに格納された業務データそのものに着目し、これを入力として業務プロセスを可視化する。

業務システムは通常、業務の進行と同時に、業務の情報を案件ごとに記録していく。例えば、受注管理システムであれば、注文を受け付けるごとに、受け付けた注文の内容とともに、その日時を記録し、一意な注文番号を採番し記録する。ここで、記録される日時はワークフローログにおける日時に対応し、注文番号はフローインスタンスの ID とみなすことができる。記録したのが注文情報のテーブルであるか生産情報のテーブルであるかというテーブルの種類は、ワークフローのタスク種別に対応する。

したがって、業務システムがデータベースに残した記録を、業務イベントの実行記録とみなすことができる。以下、業務イベントを単にイベントと記すことがある。イベントは、ID、イベント種別、実行日時という 3 つのデータの組み合わせからなる。ID とは上の例における注文番号に相当し、イベント種別はデータベースのテーブルの種類に相当する。データベースのテーブル構成がどのようになっているにしろ、この 3 つの情報が取得可能であるならば、それをイベントとみなして業務プロセスの可視化を行う。

こうした考え方に基づいて業務プロセスの可視化を実現するには技術的な課題がある。以下に課題とその解決方法を論じる。

3.1 複数テーブル

業務システムにおいては、業務データを格納するデータベースが複数存在することは一般的である。また、ひとつのデータベースの中には複数のテーブルが存在する。

業務データを入力対象とするには、この性質を前提としなければならない。ワークフローログを用いるときのように、ひとつながりのログファイルの中に全てが記録されているわけではない。

テーブルが複数ある場合には、テーブル間の参照関係を用いて、業務のつながりの情報を得る必要がある。

例えば、受注した後で発送を行い、それぞれの業務の情報が異なるテーブルに記録されている場合を想定する。このとき、受注したアプリケーションは注文テーブルに注文の品名や数量等とともに、受注日時と注文番号を記録する。また、発送業務のためのアプリケーションは、発送する品名や発送番号などともに、どの注文のための発送なのかを、注文番号によって参照して発送テーブルに記録する。こうした参照関係を用いて、業務間 (すなわちテーブル間) のつながりを追っていく。

ワークフローエンジンを用いる場合はフローインスタンス ID として業務の開始から終了まで一貫した値が振られているが、業務データを用いる場合は ID が一貫しているとは限らない。業務によってそれぞれ異なる体系の番号が採番される (上の例では注文番号、発送番号)。しかし、業務上の必要性から、後続の業務を記録するテーブルは前の業務の番号を参照している。こうしたテーブル間のキーの参照関係を用いる。

なおここで、参照関係は関係データベースの主キーと外部キーによって説明できることが多いが、BPM-E の機構上はデータベーススキーマの存在を前提としていない。データベース上の定義として外部キーが設定されている必要はないし、さらにいえばデータの格納に関係データベースが使われている必要すらない。あくまでもデータ内容そのものの分析によって参照関係を発見している。

データ間の参照関係が既知である場合には、その参照関係を BPM-E のユーザインタフェースによって人手で定義する。例えば、発送テーブルは受注テーブルを参照しており、発送テーブルの注文番号が受注テーブルの注文番号に対応する、といったことを定義する。

システム仕様が入手できないなどによってテーブル間の参照関係が明らかでない場合は、次に述べる技法によって自動的に関連付けを見出す。

3.2 関連付けの発見

分析対象のシステム(群)において、データベースのテーブル構造の知識は必ずしも入手できないことがある。著者らはそうしたときにもテーブル間の関連を自動的に推測する手法を開発した。その手法を以下に記す。

まず、テーブルの項目(列)の中の値ごとの出現回数に基づいて、項目ごとに重複度を算出する。重複度は、項目において同一値の出現回数が少ないほど小さな値をとるよう計算する。

ある項目の重複度 D の算出式は下記のように定義する。

$$D = \frac{\sum_v N_v^{\,2}}{E^2} \times 100$$

ただしここで、E はイベント数、N_v は注目している項目の値 v の出現回数を表す。ここでイベントとはテーブルの行に等しい。

次に、関連元のテーブルの項目と関連先のテーブルの項目の組み合わせを関連として生成する。この個々の関連の候補ごとに、その評価指標として、以下に説明する関連度を算出する。

関連度は関連元項目と関連先項目の組み合わせごとに算出する。関連度が 1 のときは、関連元と関連先とが漏れや重複なく対応することを意味する。関連度が 1 より小さいときは、関連元に対して関連先の一致が少ない。関連度が 1 より大きいときは、関連元に対して関連先の重複が多いことを意味する。

関連候補として関連元項目と関連先項目とを選択したとき、関連元項目の重複度が所定の重複度閾値未満であるか否かを調べる。重複度閾値は例えば 30 などと決めておく。重複度が重複度閾値以上であるときは、その関連候補は対象外として、次の関連候補に移る。重複度が重複度閾値未満のときは、次に示すように関連度を求める。

関連元項目から関連先項目への関連度 R の算出式は下記のように定義する。

$$R = \frac{\sum_v N_v M_v}{E}$$

ただしここで、E は関連元項目のイベント数、v は関連元項目の値、N_vは関連元における v の出現回数、M_vは関連先における v の出現回数を表す。

続いて、重複度と関連度に基づいて、関連の確からしさを示す関連スコアを求める。関連スコアは、関連度が 1 に近いほど大きく、また、関連元項目と関連先項目の重複度が高いほど小さくなるよう計算する。この関連スコアは後の関連ツリースコアの算出に用いる。関連スコアの計算方法の一例を下記の Ruby 言語のコード例に示す。ただしここで、関連スコアを返却する関数 rel_score の引数 rel_degree は関連度、from_dup は関連元項目の重複度、to_dup は関連先項目の重複度、key_dup は関連元テーブルのキー項目の重複度を意味する。

```
def rel_score(rel_degree, from_dup, to_dup, key_dup)
  total = rel_degree
  if total <= 1
    total = total * 100
  else
    total = 100 – total
  end
  return total – from_dup – to_dup – key_dup * 10
end
```

次に、複数の関連候補を木構造に結びつけた関連ツリーを生成する。1 つの関連は 2 つのテーブルを結びつけるものであるが、最終的に求めたいのは業務の開始から終了までをカバーする関連のつながりである。関連のつながりは木構造で表現できるので、関連ツリーと呼ぶ。関連のつながりが直線的なリスト構造でなく木構造になるのは、あるテーブルを参照するテーブルが 2 つ以上あり得るためである。

関連ツリーは可能なものが複数あり得るので、関連ツリー候補として有力なものをいくつか利用者に提示し、最終的には利用者が候補の中から最も適切な関連ツリーを選択するという手順を踏む。

ひとつの関連ツリー候補に対しては、関連ツリースコアが算出される。関連ツリースコアは、関連ツリー候補に含まれる関連の関連スコアの合計である。関連ツリースコアの大きな関連ツリー候補ほど、関連ツリーとしての確からしさが高いことを意味する。

以上により、関連ツリースコアの高い関連ツリー候補を見つけるという形で、テーブル間の関連付けを発見する。

なお、ここに挙げた方法はデータの参照関係が未知の場合だけでなく、参照関係の仕様が入手可能な場合にも役に立つ。仕様が入手可能なときは人手で関連付けを定義するが、関連付けのユーザインタフェースにおいて最も確からしい関連付けを上位に提示することで使いやすくするといったことが、この手法によって可能となる。

3.3 フローインスタンスの生成

上記の方法によってテーブル間の関連付けが発見されると、その構造に従って個々のフローインスタンスのデータを得ることが可能となる。

例として、図 2 の業務データベースのテーブル群を想定する。

受注テーブル

日時	受注番号	地域	担当
6/1 9:01	A01	東京	A
6/2 9:35	A02	東京	A

生産テーブル

日時	生産番号	受注番号	品番	納期
6/2 10:22	S01	A01	P01	6/6
6/5 15:05	S02	A01	P02	6/10
6/6 11:41	S03	A02	P01	6/10

手配テーブル

日時	手配番号	受注番号	品番	納品先
6/10 9:50	T01	A01	P01	東京
6/11 9:05	T02	A01	P02	東京
6/11 9:10	T03	A01	P02	東京
6/12 8:55	T04	A02	P01	東京

配送テーブル

日時	手配番号	配送便	納品先
6/19 11:23	T01	H01	東京
6/19 12:01	T02	H01	東京
6/19 12:10	T03	H01	東京

図 2　業務データのテーブル例

このテーブル群に対して、関連付けとして図 3の構造が分かったとする。ただしこの図で矢印は関連元から関連先へと引かれている。

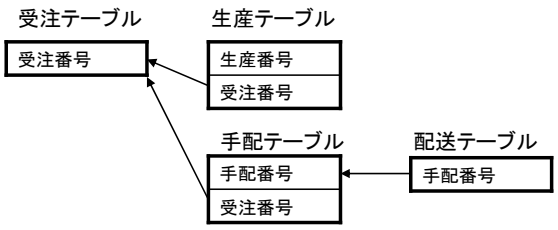

図 3　テーブルの関連

このとき、フローインスタンスを構成するイベントインスタンスとして、受注テーブルの受注番号 A01 の行から出発すると、関連を逆にたどって、生産テーブルの生産番号 S01, S02 の行、手配テーブルの T01, T02, T03 の行、配送テーブルの T01, T02, T03 の行が、それぞれ得られる。

これらの行 (すなわちイベントインスタンス) を、日時の順に整列すると、1 つの完結したフローインスタンスとなる。

こうして業務データの中から全てのフローインスタンスを抽出すると、次節に記す方法で業務フロー図を生成することができる。

4. 業務フロー図の表示

前節の技法により業務システムのデータから業務フローインスタンス群を抽出することが可能となった。

抽出された業務フローをフロー図として可視化するにあたり、モデルの推定と、実態の分析という、2 つの異なる考え方がある。

4.1 モデルを推定することとの違い

モデルを推定するという概念では、技術のゴールは、出力されたログをうまく説明できるような業務プロセスモデルを推定することに設定される。2 節に示した関連研究の多くはこうした考えに基づいている。この考え方では、例外的なログはノイズとみなしてこれを除去し、綺麗なモデルを導くことが重要となる。

一方、著者らのアプローチは、実態の分析を主眼とし、モデルの推定は行わない。フロー図に綺麗でない結果をもたらす例外的な業務履歴は、排除すべきノイズではなく、業務ないしシステムの改善に結び付けるべき、積極的な分析対象であるというスタンスを取る。

このようなアプローチを取る理由としては、利用者が興味を持つのはあくまでも実態そのものであって、アルゴリズムによって推定されたモデルではないことが多いということが挙げられる。著者らの経験上、業務プロセスの可視化によって想定外の業務フローを発見した利用者は、個別具体的な業務の流れを分析する傾向にある。こうした目的には、計算機によって自動的に抽象化されたモデルよりも、具体的な事実そのものを分析する技術が有用である。

4.2 出現頻度による典型/例外の分離

抽出されたフローインスタンスを視覚化するには、業務イベントをノード、イベント間の線をエッジとする有向グラフとして表現する。複数のフローインスタンスを可視化するには、全フローインスタンスの情報を重ね合わせて、重複するノードとエッジを集約して表現する。この簡単な例を図 4に示す。図 4上段の 2 本のフロー種別を重ね合わせると、下段のフロー図が得られる。なおこの図で、1 つのイベントから 2 つのイベントに矢印が引かれているのは、実行時に何らかの条件によっていずれかに分岐することを意味する。記法は異なるが、BPMN [11]の排他ゲートウェイと同様である。

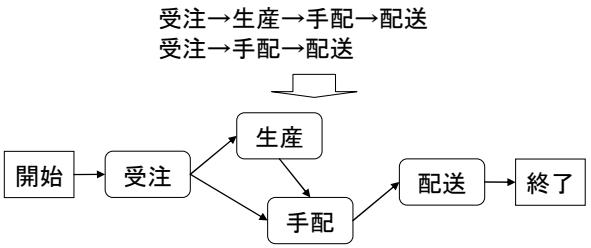

図 4　フロー種別を重ね合わせてのフロー図描画

この方法で業務フロー図を描画することはできるが、しかし問題がある。業務データから抽出された全てのフロー種別を重ね合わせて可視化すると多くの場合、Günther ら [7] が「スパゲッティ」と表現するように、人が見て理解できない複雑すぎるフロー図となるのである。

提案手法においては、フロー種別をその出現頻度に応じて典型フローと例外フローとに分離し、典型フローのみを表示することによって、この問題に対処する。

具体的には、まず、フロー種別をその出現回数によって整列する。ここでフロー種別とは、フローを構成するイベントの列によって定義される。イベント種別が同じ順序で出現するフローは同一のフロー種別であるとみなす。繰り返しの有無も含めて 1 箇所でも構成イベントが異なれば異なるフロー種別とみなされる。例えば、A→B→C と A→B→B→C とは異なるフロー種別である。

その上で、任意の比率を典型フロー比率として設定する。例えば 0.8 とする。この場合、全フローインスタンスのうち頻度の最も高い方から上位 80%を典型フローとして採用することを意味する。全フローインスタンス数をこの典型フロー比率に乗じた数を閾値として設定する。

次に、出現数の最も多いフロー種別から典型フロー群に採用していき、該当フロー種別の出現数をフローインスタンスの累積数に加算する。これを繰り返し、フローインスタンス数の累積数が閾値に達するまで、フロー種別を典型フロー群に採用していく。

こうして得られたフロー種別群を用い、重ね合わせてフロー図を描画することで、例外的な流れを取り除いた、典型的な業務フロー図が得られる。

この方法によって、多くの場合に、人が見て理解しやすい業務フロー図となる。この方法が基づいている原理は、業務プロセスにおいて、多数のフローインスタンスが少数の典型的なフロー種別によって占められ、それ以外は少数のインスタンスしか持たない多様なフロー種別からなるという性質である。この性質は定型処理を前提とする業務において自然なことである。非定型な処理においてはこの性質は当てはまらないが、非定型な業務はそもそも BPM に向いていない。

なお、この手法において、典型フロー比率は事前に決定できるわけではなく、可視化結果に基づいてフロー図が適当な複雑さになるような比率に調整する必要がある。これは GUI から人手によって行うことも容易であるし、またフロー図の複雑さをプログラムが評価して適切な典型フロー比率を決定してやることも可能である。フロー図の複雑さの評価としては、フロー図のノードの次数が一定の値以下になるように典型フロー比率を調整するという方法が例えば可能である。

本手法の良い点は、「スパゲッティ」にならない見やすい表示を実現しながらも、抽象化や推定ではなくあくまで現実に発生した業務の流れを可視化していることである。モデルでなく実際に起こったことに関心を持つ利用者にとっては、この特徴は大きな価値を持つ。

5. 例外フローの分析

5.1 例外フローに着目する意義

前節までにおいて典型的な業務フローを可視化することができるようになった。しかし、実用的な観点からは、典型的な業務フローを可視化するだけでは不十分である。

なぜなら、典型業務フローは業務運用者にとって多くの場合、既知の知識だからである。業務が典型的にどう流れているかは業務プロセスに責任を持つ人は当然知っている筈のことである。新たな知見を産み出すためには、数の少ない例外的なフローに注目するのが効果的である。そうした例外的なフローには、業務上あるいはシステム上の問題が潜んでいる可能性が高い。業務あるいはシステムを改善するためには典型フローだけでなく例外フローも分析することが有用である。

例外フローとは、本手法においては、全業務フローのうち、典型フローでないものである。典型業務フロー図から排除された例外フローを分析する方法を次に述べる。

5.2 例外フローの探し方

例外フローは、フロー種別ごとのインスタンス数が少なくかつ種類が多いという特徴を持つ。このため、典型フローと同じように流れを重ね合わせると、意味の取れない複雑な図となってしまう。

例外フロー群から意味のある知見を得るためには、なんらかの特徴を持つフロー種別を探し当てることが必要になる。本手法において業務フローとは業務イベントの並びであるので、イベントの並び方の中に特徴を見出して、問題のありそうなフローを抽出することとなる。

ここでは、繰り返しの多いフロー、手戻りの多いフロー、規則に合致しないフローという、3 つの効果的な抽出法について述べる。

まず、第 1 の、繰り返しの多いフローとは、1 つのイベントを何度も繰り返し実行しているステップを含むフローである。繰り返しを含むフローの例を図 5に掲げる。この図の例は「手配」を繰り返し実行することを示す。同じタスクを何度も実行していることは、何らかの非効率性の現れである可能性が高いと考えられる。

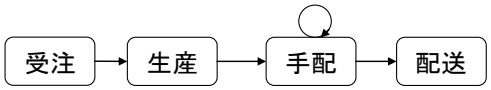

図 5　繰り返しのあるフローの例

業務フローの中から繰り返しの多いフローを発見するには、フローを構成するイベントの並びを見て、同じイベントが連続して出現する回数を数え上げる。この回数を例外フロー種別ごとの得点として保持し、この得点の大きな例外フロー種別を抽出する。

次に、第 2 の、手戻りの多いフローとは、業務の流れの中で、過去に実行したタスクに遡って再び実行している部分を含むフローである。手戻りを含むフローの例を図 6に掲げる。この図の例は「手配」を実行した後で前の「生産」に再び戻ることを示す。前のステップへの手戻りは、何らかの非効率性の現れである可能性が高いと考えられる。

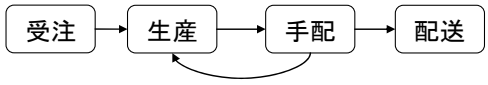

図 6　手戻りのあるフローの例

業務フローの中から手戻りを発見する方法を Ruby 言語によって以下に示す。メソッド count_return の引数 event_sequence はフローを構成するイベント種別の列であり、戻り値はそのフローが含む手戻りの数である。

```
def count_return(event_sequence)
 count = 0
 p = 0
 list = []   # 空の配列を用意
 event_sequence.each do |event|
  if list.include?(event)
   position_of_event = list.index(event)
   if position_of_event <= p
    count += 1
   end
  else
   list.push(event)   # list の末尾に追加
  end
  p = list.index(event)
 end
 return count
end
```

このロジックの概要は以下のとおりである。最初に、同種のイベント種別を 1 つしか含むことのできない正規イベント列を、空の列として初期化する。また、注目位置を示すポインタを初期化する。そのうえで業務フローを構成するイベント列を先頭から見ていく。注目しているイベントが正規イベント列に存在しないならば、正規イベント列に注目イベントを追加する。存在するならば、これは手戻りの可能性を意味する。注目イベントがポインタ位置より小さいならば、手戻りがあったことになり、手戻り回数を 1 増やす。注目イベントが正規イベント列にあった場合もなかった場合も、ポインタ位置を、正規イベント列の中で注目イベントが出現する位置に設定し、次のイベントへ移る。

この手続きによってフロー種別ごとの手戻り数を求めることができる。手戻り数の大きな例外フローを利用者に提示することで、業務の中の非効率な手続きを発見する助けとすることができる。

次に、第 3 の、規則に合致しないフローとは、定義されている業務規則からの逸脱が存在するフローである。例えば、あるタスクを実行するのに、事前に実行しておかなければならないタスクが存在するとする。にもかかわらず、事前に実行すべきタスクを経ることなしに該当タスクを実行しているというフローには問題がある。具体例としては、購入の前に決裁が必要なのに、決裁せずに購入を実行しているといったことが挙げられる。この例を図 7に示す。

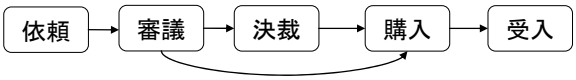

図 7　規則に合致しないフローの例

こうした問題のあるフローを発見するには、最初に規則を定義する。規則とは例えば、「購入」イベントを実行する前に「決裁」イベントを実行していなければならない、といったことである。

その上で、個々の例外フローについて、規則を満たしているかどうかを判断する。判断のためには、あるフロー種別が「購入」を含むならば、その前に「決裁」が実行されているかどうかを調べる。実行されていないフローは、規則を満たしていない。規則を満たしていないフローを利用者に提示する。

これにより、業務規則に従っていない運用を発見することが可能となる。

5.3 例外フローの提示

このようにして抽出された例外業務フローは、単にそのフローのみを描画するよりも、典型業務フローに重ね合わせて描画することで、典型的な業務実行と比較考量することができる。これにより、例外フローの問題がどこにあるかの理解を助けることができる。図 8に、例外フローを典型フロー図に重ね合わせる例を示す。ここでは、典型フローと重ね合わせることによって、必要な「決裁」処理がとばされていることが見て取れる。

図 8　例外フローを典型フローに重ね合わせる例

典型フロー図は個々のフローが持つイベント間の遷移を集約して図示するものであるが、そこに重ね合わせる例外フローは、遷移を典型フローのそれとは集約せずに別のエッジとして、点線など典型フローと区別できる形で出力する。こうすることで典型的な流れとの違いを利用者が認識しやすくなる。図 8においては、実線で典型フローを、破線で例外フローを示している。

ここに述べた例外フローを発見・提示する機構により、業務上問題のある運用、あるいはシステム上の問題のある箇所が、業務プロセスの責任者によって特定しやすくなる。

6. 事例

前節までに説明した BPM-E の技法を実際の業務システムに適用した事例 2 件を以下に示し、有効性を検証する。

6.1 製造業のプロセス

図 9は、ある製造業における業務プロセスを本手法により可視化した業務フロー図である。図 9においては抽出された全フロー種別を描画している。このフロー図は明らかに理解が不可能である。この図で塗りつぶされているように見えるのは、多数の線が重なっているものである。

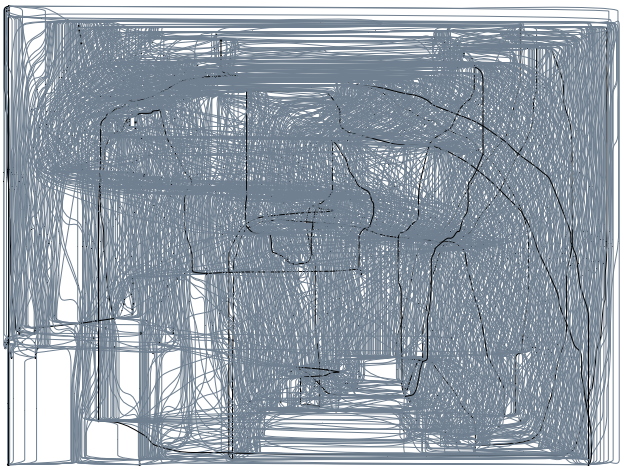

図 9　ある製造プロセスの全フローインスタンス

一方、同じデータを用いて、全フローでなく、出現頻度上位 40%の典型フローを描画した業務フロー図が図 10である。このフロー図は多数の例外フローが除去されており、十分に目視に耐え得る。

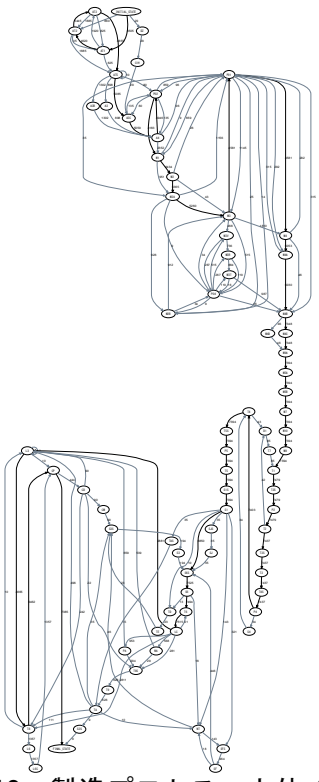

図 10　製造プロセス、上位 40%

この典型フローを分析すると、3 つの段階に対応することが分かった。最初のやや複雑な部分は準備工程、それに続く直線的な部分は製造工程、最後の入り組んだ部分はテスト工程である。

最後のテスト工程においては他の工程よりも複雑でイベント間の行き来が多くなっているのが見て取れる。

この可視化結果から、業務改善案として、テスト工程に対して業務の標準化を進めていくことで効率化を図るという方針が導かれた。

ここでは、出現頻度の高い典型的なフローに限定することで理解しやすくする例をとりあげた。次の事例では、例外フローの分析の例を見る。

6.2 稟議システムのプロセス

図 11は、ある企業の稟議システムの業務を可視化した典型フロー図である。上位 60%のものである。図中、イベント間の遷移の横に付けられている数字はその遷移の発生した件数である。

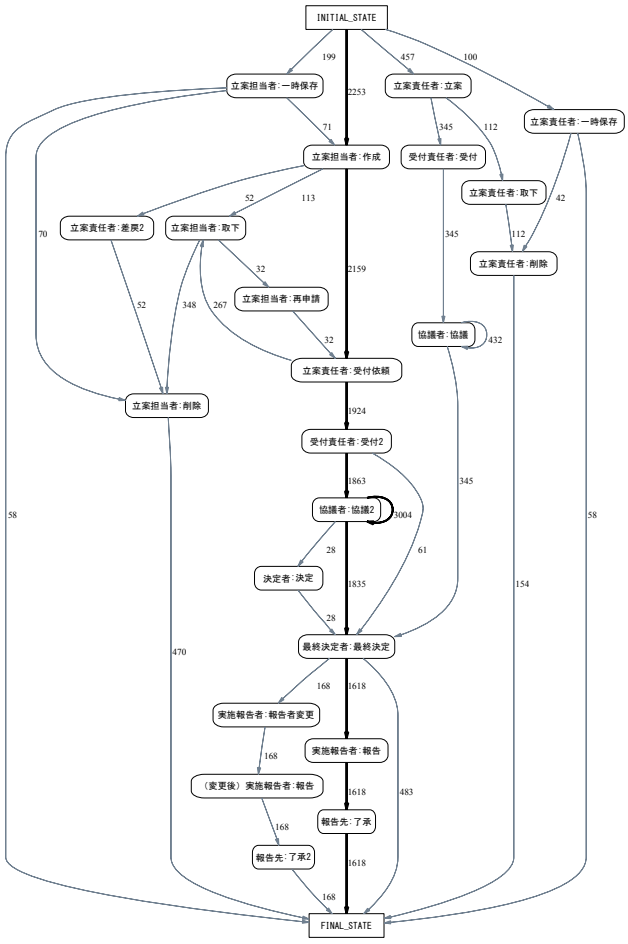

図 11　稟議プロセス、上位 60%

この業務に対し、例外フローの分析を試みた。

繰り返しの多いフローの抽出によって、図 12に示す箇所が得られた。この図も含め、例外フローを示す図には、例外フローの遷移を破線で示すとともに、当該例外フローの実行順序を示す連番を、括弧書きの数字として矢印の脇に記している。これは繰り返しや手戻りがあるときに実行順序を把握できるようにするための措置である。

この図は「協議者:協議 2」というタスクを何回も繰り返し実行していることを意味する。この何度も繰り返されている箇所が、業務改善の手がかりとなる。この結果に基づいて、協議を何度も行わなくても済むようにコミュニケーションを改善するという、業務改善の指針が導かれた。

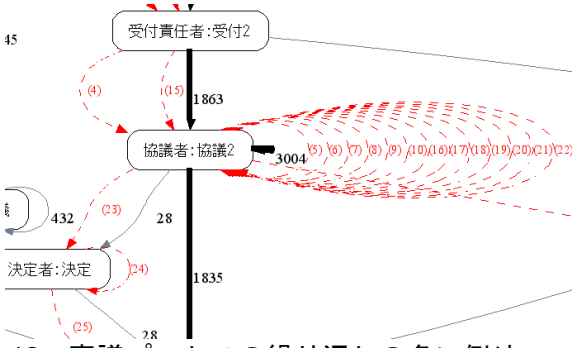

図 12　稟議プロセスの繰り返しの多い例外フロー(部分)

次に、手戻りの多い例外フローの抽出を行ってみた。これによって得られた例外フローの拡大図を図 13に示す。

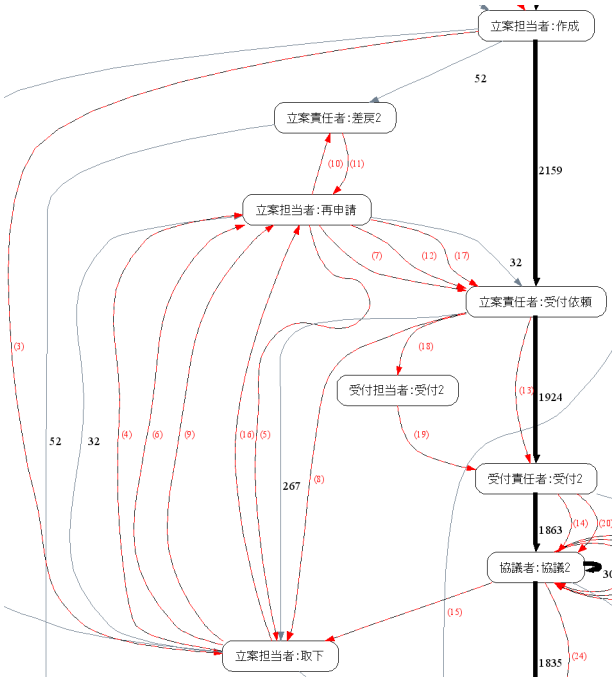

図 13　稟議プロセス、手戻りの多い例外フロー(部分)

この例外フローにおいては、「立案担当者:取下」と「立案担当者:再申請」を何度も行ったり来たりしていることが見て取れる。こうした例外フローを発見したならば、このフローのインスタンスに関連づけられている業務データを調査することで、どのような案件でこうした多数の手戻りが発生しているのかを特定することが可能となる。

次に、規則に合致しないフローの抽出によって、図 14に示す例外フローが得られた。本来は、「実施報告者:報告者変更」のタスクを実行してから「(変更後)実施報告者:報告」のタスクを実行しなければならない規則になっている。典型フロー(実線矢印)では実際そのように運用されているが、この抽出された例外フロー(破線矢印)では「実施報告者:報告者変更」を経ずに「(変更後)実施報告者:報告」を実行している。これは規則に照らして正当でない運用がなされていたことが本手法によって明らかにできるということ

を意味する。本事例は単なる社内の運用規則でしかなく深刻さはさほどでないが、もし法令遵守にかかわるケースだとすると重要性は非常に高いことになる。

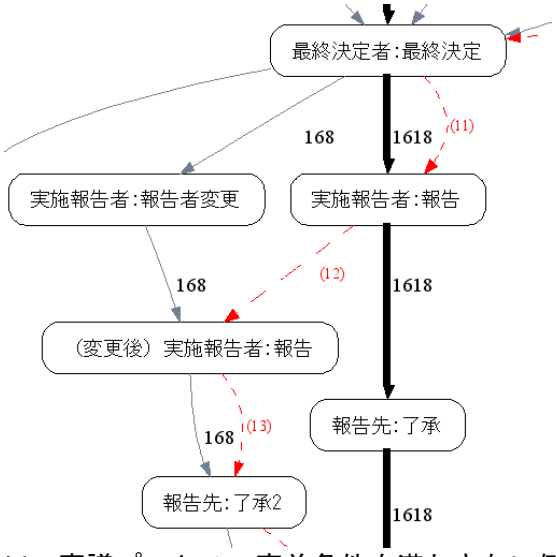

図 14　稟議プロセス、事前条件を満たさない例外フロー(部分)

7. おわりに

本論文ではワークフローのログではなく業務システムのデータそのものを使って業務プロセスの実態を可視化する技法を述べた。可視化するにあたっては理想的なモデルを推定するのではなく、あくまでも実態の分析に資する手法として、フローの出現頻度に応じて典型フローと例外フローとに分類して理解しやすい業務フロー図を描画する方法を示した。単に典型的なフローを示すだけでなく、例外フローを分析することで、業務やシステムの問題点を発見することに役立てることができる。例外フローの効果的な抽出法を 3 種類示した。

本手法では並行分岐などは扱っていない。これはモデルの推定に属する技術となり、関連研究に示した既存研究を役立てられる可能性がある。また、本論文ではイベントの発生日時として 1 つの日時のみ存在するようなケースを専ら扱ったが、開始と終了の両方の日時が入手可能なケースにおいては、イベント実行時間のオーバーラップなどを考慮することによって実態としての並行実行を可視化することが可能になると考えられる。

参考文献

[1] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, A.J.M.M. Weijters, “Workflow mining: A survey of issues and approaches”, Data & Knowledge Engineering, Volume 47, Issue 2 (2003).

[2] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster, “Workflow Mining: Discovering Process Models from Event Logs”, IEEE Transactions on Knowledge and Data Engineering, Volume 16, Issue 9 (2004).

[3] W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, H.M.W. Verbeek, “Business Process Mining: An Industrial Application”, Information Systems, Volume 32, Issue 5 (2007).

[4] R. Agrawal, D. Gunopulos, and F. Leymann, “Mining Process Models from Workflow Logs”, Sixth International Conference on Extending Database Technology (1998).

[5] J. Cook, A. Wolf, “Automating process discovery through event-data analysis”, Proc. 17th Intl. Conf. on Software Engineering (1995).

[6] B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, W.M.P. van der Aalst, “The ProM Framework: A New Era in Process Mining Tool Support”, Applications and Theory of Petri Nets 2005, Lecture Notes in Computer Science (2005).

[7] Christian W. Günther, Wil M.P. van der Aalst, “Fuzzy Mining -- Adaptive Process Simplification Based on Multi-perspective Metrics”, Business Process Management (2007).

[8] J. Herbst, “A Machine Learning Approach to Workflow Management”, Proceedings 11th European Conference on Machine Learning, volume 1810 of Lecture Notes in Computer Science (2000).

[9] San-Yih Hwang, Wan-Shiou Yang, “On the discovery of process models from their instances”, Decision Support Systems, Volume 34, Issue 1 (2002).

[10] R.S. Mans, M.H. Schonenberg, M. Song, W.M.P. van der Aalst, P.J.M. Bakker, “Application of Process Mining in Healthcare---A Case Study in a Dutch Hospital”, Biomedical Engineering Systems and Technologies (2008).

[11] Object Management Group, “Business Process Model and Notation (BPMN) Version 2.0”, http://www.omg.org/spec/BPMN/2.0 (2011).

[12] Shiro Ogasawara, Kenichi Tayama, Tetsuya Yamamura, “Estimating Structures of Business Process Models from Execution Logs”, Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (2006).